Hao Li

Geometry Fundamentals of Computer Vision

上海交通大學出版社
Shanghai Jiao Tong University Press

Springer

Hao Li

Geometry Fundamentals of Computer Vision

Springer

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

Hao Li
Shanghai Jiao Tong University, Shanghai, China

ISBN 978-981-96-7343-8   e-ISBN 978-981-96-7344-5
https://doi.org/10.1007/978-981-96-7344-5

Jointly published with Shanghai Jiao Tong University Press
The print edition is not for sale in China (Mainland).
Customers from China (Mainland) please order the print book from: Shanghai Jiao Tong University Press.

© Shanghai Jiao Tong University Press 2025

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

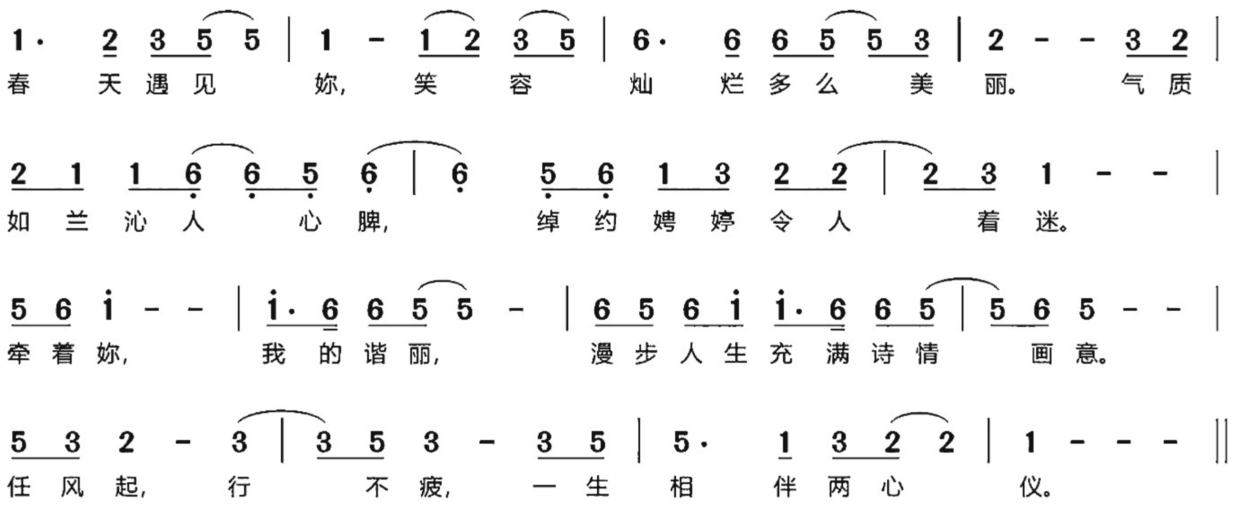
The publishers, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publishers nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publishers remain neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

À ma chérie ***Fangqi***
Quand je passe devant le champs de fleurs,
je n’ai pas envie de tourner ma tête;
grâce à la poursuite de la vérité d’une part
et à vous d’autre part

1 = C $\frac{4}{4}$

李颢 词
李颢 曲


春天遇见妳，笑容灿烂多么美丽。气质
如兰沁人心脾，绰约娉婷令人着迷。
牵着妳，我的谐丽，漫步人生充满诗情画意。
任风起，行不疲，一生相伴两心仪。

Preface

Computer vision is nowadays a popular concept, thanks to ubiquitousness of computer vision related applications and public media’s wide broadcasting of them in the society, such as face detection and fingerprint recognition in the context of intelligent offices, lane detection, traffic sign detection, and pedestrian detection in the context of intelligent vehicles, cereal grain inspection in the context of intelligent agriculture, license plate location and human gait analysis in the context of intelligent surveillance, video games and virtual reality in the context of entertainment. Even sometimes when I take taxi no matter in China or in Europe, the taxi driver may chat a lot about computer vision.

Nowadays, the Internet is maturely developed. Most knowledge in typical computer vision references and open-source code such as the famous *OpenCV*[1] are easily accessible on the Internet. If readers are just interested in having available code to fulfil certain computer vision task and are probably interested as well in acquiring some relevant knowledge explaining the specific code, they can fairly take advantage of the Internet only, without resorting to any book on computer vision. Then what is the motivation of writing this book?

Writing a book on *Geometry Fundamentals of Computer Vision* is motivated directly by the author’s years of teaching activities with his teaching collaborators from Mines ParisTech in the course computer vision at Shanghai Jiao Tong University (SJTU). The course mainly consists of two parts: the part of geometry fundamentals, and the part of machine (especially deep) learning fundamentals. This book focuses right on the former part and aims at providing a systematically organized presentation of geometry fundamentals of computer vision for students as well as field engineers.

Besides above motivation that stems from teaching activities at Shanghai Jiao Tong University, other motivations stem from two existing “surprises” in the author’s personal opinion. The first “surprise” is that in the latest dozen of years, i.e. since the renaissance of deep learning, a lot of technique men are too used to artificial neural networks especially deep networks (or often shortly called networks), and tend to rely on networks blindly. Proposals like “using these or those networks to tackle ...” are often heard, whereas mature methods, which are not based on networks but utilize geometry fundamentals of computer vision, can effectively solve the concerned problems already.

Somewhat related to the first “surprise”, the second “surprise” is that a lot of technique men tend to treat computer vision as a kingdom of empirical knowledge and do not care (or know) essential reasons, i.e. why behind computer vision methods. Negligence of essential reasons behind computer vision methods might be indirectly due to the renaissance of deep learning, as a lot of technique men are too used to enjoy the black box style convenience brought by networks and have no mood to reflect why.

However, the author would like to point out that geometry fundamentals of computer vision are indeed valuable knowledge other than that of deep learning or networks for computer vision. This is why the previously mentioned course computer vision at SJTU consists of the part of geometry fundamentals as well as the part of machine learning fundamentals. The author would also like to argue that possessing the spirit of *know-why* is beneficial to work on computer vision, especially in the long run. Throughout this book, the author tends to provide detailed theoretical derivations and explanations to clarify essential reasons behind computer vision methods.

The author also provides a large amount of original demonstration code in Matlab scripts. The author believes that complete code demonstration is especially important and even indispensable to strengthening readers' understanding of knowledge presented in this book and to enhancing their ability to flexibly apply relevant knowledge in practice, though some Matlab built-in functions are indeed available. The reason is that for geometry fundamentals of computer vision, the gap between how humans comprehend relevant knowledge and how computers can effectively put corresponding logic into practice does exist inevitably. Complete code demonstration aims right at bridging such gap, clarifying algorithmic implementations of various computer vision methods for practical applications.

The author would like to dedicate this book to his mother **Lianhua Zhan** for her priceless contribution to family education and family building during the author's childhood. The author would also like to dedicate this book to his father **Jinhong Li** for his mechanical engineering expertise which directly attracted the author into the wonderful kingdom of engineering especially on automatic systems and robotics. The author would like to express his sincere gratitude and appreciation to his colleagues at Shanghai Jiao Tong University (SJTU) for their encouragement to his teaching activities, to his former colleagues especially Arnaud de la Fortelle, Fawzi Nashashibi, Michel Parent, Christian Laugier, Fabien Moutarde, Bruno Steux, Claude Laurgeau at Institut National de Recherche en Informatique et en Automatique (INRIA) and Mines ParisTech for their encouragement and help to his academic life, to his dear friends in Sophia-

Antipolis and Côte d’Azur for their precious friendship in his life.

Hao Li
Shanghai, China
2024

Footnotes

[1] OpenCV official website: https://opencv.org/.

Contents

1 Introduction

- 1.1 Vision Systems
- 1.2 Image Representation and Two-Dimensional Geometry Analysis
- 1.3 Computer Vision and Three-Dimensional Geometry Analysis
- 1.4 Scope Preview
- 1.5 Code Demonstration for Practice
- References

2 Image Processing for Feature Extraction

- 2.1 Image Coordinates System
  - 2.1.1 From Discrete Indices to Continuous Coordinates
  - 2.1.2 Linear Interpolation
  - 2.1.3 Quadratic Interpolation
  - 2.1.4 Image Resolution Enhancement
- 2.2 Feature Points
  - 2.2.1 Thresholding: Pixel-Wise Point Extraction
  - 2.2.2 Application of Image Thresholding: Watermark Removal
  - 2.2.3 Adaptive Thresholding
  - 2.2.4 Non-Maximum Suppression: Extremum Point Extraction
  - 2.2.5 Dilation and Erosion: Contour Extraction

2.3 Edge Points
    2.3.1 Gradient Image
    2.3.2 Directional Non-maximum Suppression
    2.3.3 Double Thresholding
    2.3.4 Canny Edge Detection
2.4 Corner Points
2.5 Scale-Space Feature Extraction
    2.5.1 Scale Space: Filtered Images Associated with Various Filtering-Scale Parameter
    2.5.2 Gaussian Pyramid
    2.5.3 Scale-Invariant Laplacian
    2.5.4 Difference-of-Gaussian
2.6 Scale Invariant Feature Transform (SIFT)
    2.6.1 Feature Point Localization
    2.6.2 Local Image Orientation Assignment
    2.6.3 Local Image Descriptor Generation
2.7 Post-SIFT Features
    2.7.1 Fast Filtering: Example of SURF
    2.7.2 Fast Feature Detection: Example of FAST
    2.7.3 Simplified Feature Description: Example of BRIEF
    2.7.4 Oriented FAST and Rotated BRIEF (ORB)
2.8 Machine Learning Based Features
    2.8.1 Pre-Deep Learning Methods
    2.8.2 Deep Learning Methods

2.8.3 Learn for Feature Extraction
2.9 Note
References

3 Computational Geometry
3.1 Distance
3.1.1 Euclidean Distance
3.1.2 Minimum-Distance-Path Perspective
3.1.3 Point-to-Line Distance
3.1.4 Point-to-Plane Distance
3.2 Generalized Distances and Measures
3.2.1 L_p-Norm Distance
3.2.2 City-Block Distance
3.2.3 Mahalanobis Distance
3.2.4 Distance, Area, Volume, and Fractal Measures
3.3 Graph
3.3.1 Topological Abstraction
3.3.2 Planar Graphs and Euler Formula
3.3.3 Planar Straight-Line Graphs
3.3.4 Point-Location Problem
3.4 Convex Geometry
3.4.1 Convex Polygon
3.4.2 Point-Inclusion Problem
3.5 Closest Point Problems

        - 3.5.1 Find the Closest Pair Among Points
        - 3.5.2 Voronoi Diagram
    - References
- 4 Projective Geometry and Camera Model
    - 4.1 Projective Geometry
    - 4.2 Camera Model
        - 4.2.1 Pinhole Camera Model
        - 4.2.2 Extrinsic and Intrinsic Parameters
        - 4.2.3 Perspective Mapping
        - 4.2.4 Revisit Common Sense of Projective Geometry
    - 4.3 Homography
        - 4.3.1 Bijective Mapping Between Two-Dimensional Planes
        - 4.3.2 Homography Calibration
        - 4.3.3 Inverse Perspective Mapping
    - 4.4 Image Distortion
        - 4.4.1 Distortion-Considered Camera Model
        - 4.4.2 Inverse Distortion Model and Distortion Rectification
    - 4.5 Stereo Vision
        - 4.5.1 Why Stereo Vision Is “Natural”?
        - 4.5.2 Epipolar Constraint: Fundamental and Essential Matrices
        - 4.5.3 Rectified Stereo Vision

4.5.4 Binocular Disparity

References

5 Camera Calibration

5.1 Camera Intrinsic Calibration

5.1.1 Exploitation of Known Homography

5.1.2 Chessboard Pattern for Camera Intrinsic Calibration

5.1.3 Initial Estimate of Camera Intrinsic Parameters

5.2 Refined Estimate of Camera Intrinsic Parameters

5.2.1 Three-Dimensional Rotation and Rodrigues Formula

5.2.2 Lie Group Interpretation of Three-Dimensional Rotation

5.2.3 Estimate Refinement via Optimization

5.3 Image Distortion Calibration

5.4 Camera-X Calibration

5.4.1 Fundamental Matrix Calibration

5.4.2 Essential Matrix Calibration

5.4.3 Stereo Camera Calibration

5.4.4 Camera-Range Sensor Calibration

References

6 Pattern Recognition for Feature Organization

6.1 Clustering

- 6.1.1 Direct Connection
- 6.1.2 K-Means
- 6.1.3 Mean Shift
- 6.1.4 Normalized Cut
- 6.1.5 Segmentation

6.2 Known Feature-Model Relationship Oriented Model Fitting

- 6.2.1 Least Mean Squares (LMS) Method
- 6.2.2 Recursive Linear LMS
- 6.2.3 Regularized LMS

6.3 Unknown Feature-Model Relationship Oriented Model Fitting

- 6.3.1 Hough Transform
- 6.3.2 Expectation Maximization (EM)
- 6.3.3 Random Sample Consensus (RANSAC)

References

7 Applications

7.1 Ground Object Detection

7.2 Virtual Reality and Augmented Reality

7.3 Cooperative Augmented Reality: See-Through Effect

7.4 Image Stitching

7.5 Special Tag Based Visual Localization

7.6 Visual SLAM

References

Appendix A: Integral Image
Appendix B: Illusion of Brightness
Appendix C: Image Filtering
Appendix D: Curves and Surfaces
Appendix E: Scale-Space Theory
Appendix F: Equations
Appendix G: Artificial Neural Networks
Index

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_1

1. Introduction

Hao Li[1] ✉
(1) Shanghai Jiao Tong University, Shanghai, China

1.1 Vision Systems
1.2 Image Representation and Two-Dimensional Geometry Analysis
1.3 Computer Vision and Three-Dimensional Geometry Analysis
1.4 Scope Preview
1.5 Code Demonstration for Practice
References

1.1 Vision Systems

Vision plays an indispensable role in the world. There is a famous figurative saying: “Eyes are windows of mind”. This saying highlights the importance of one’s eyes namely vision organs which contribute to more than eighty percent of one’s acquired information from this world.

Vision itself would not be of such huge importance if one did not have a powerful brain that can process visual information effectively. One’s eyes need towork together with one’s brain; in other words, one’s eyes and brain form a holistic vision system. Such holistic vision systems formed by human eyes and brains are called **biological vision systems**.

The development of technology leads to the birth of digital cameras and computers (or digital processors) that can be compared to human eyes and brains respectively. By analogy, digital cameras and computers, which are digital counterparts of human eyes and brains, can also form another kind of holistic vision systems. Such holistic vision systems formed by digital cameras and computers are called **computer vision systems**.

A computer vision system can be simply regarded as combination of a vision sensor (usually a camera) and a computer, as illustrated in Fig. 1.1. Basically speaking, any works that involve using a vision sensor to capture images and then processing captured images by computer can be categorized as works of computer vision. So the kingdom of computer vision is extremely wide, involving a number of important domains such as the domain of image processing [1] and the domain of pattern recognition [2] that are already wide themselves.

However, this book focuses on **geometry fundamentals of computer vision**. For this purpose, the author will present image processing methods that are intended for extracting geometric entities such as points, edges, and corners from images. The author will present pattern recognition methods that are also intended for obtaining and organizing geometric entities such as shapes and structures in images. In other words, the author will present knowledge of image processing and pattern recognition that is intended for two-dimensional geometry analysis of images.

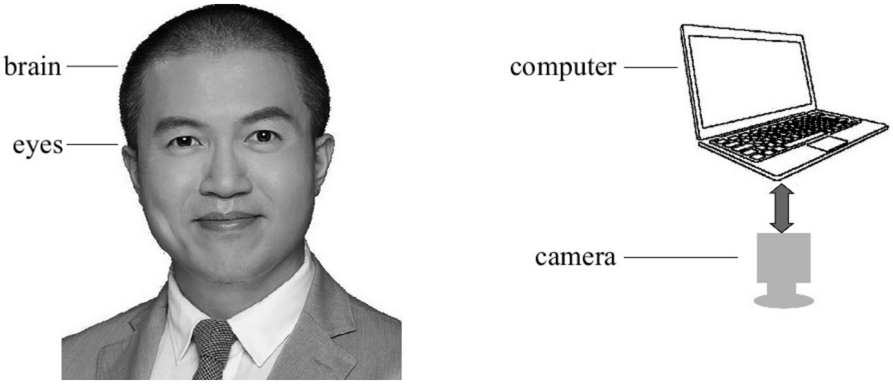


Fig. 1.1 Biological vision systems and computer vision systems

Not limited to image processing and pattern recognition for two-dimensional geometry analysis of images, the author will further present other core knowledge of computer vision, especially knowledge on visual relationship between real world entities and their projection counterparts in images. Such knowledge of computer vision includes knowledge on projective geometry, camera model, homography, camera calibration, three-dimensional rigid transform and so on. Besides, the author will also present pattern recognition methods that are intended for obtaining and organizing three-dimensional geometric entities among images. In other words, the author will present knowledge of computer vision that is intended for three-dimensional geometry analysis of images. The author will also present how to take advantage of such knowledge flexibly to realize various kinds of visual effects that are actually technology basis of many interesting and useful applications.

1.2 Image Representation and Two-Dimensional Geometry Analysis

Images form a fundamental concept of computer vision. Nowadays, digital devices including digital cameras are ubiquitous, whereas analog cameras are rarely used. A **digital image** is normally obtained as follows: A camera captures the physical image of world entities and projects the physical image onto a photosensitive plane. Then **vision discretization** takes place, which includes two aspects of discretization: one is **space discretization** where the entire physical image is split onto a matrix of perceptive cells with distinguishable positions on the photosensitive plane, and the other is **value discretization** where each perceptive cell is assigned a discrete value corresponding to the light intensity of the physical image fraction projected onto the perceptive cell.[1] Vision discretization is illustrated in Fig. 1.2.

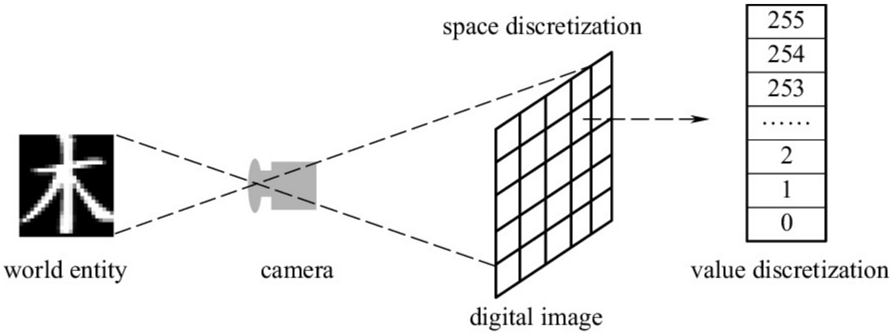
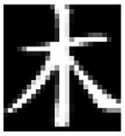


Fig. 1.2 Vision discretization to obtain digital images



0	0	0	0	0	0	0	0	0	0	141	255	255	192	43	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	34	238	255	255	128	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	178	255	248	54	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	141	255	248	54	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	141	255	243	22	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	98	255	243	22	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	98	255	243	22	0	4	62	140	153	153	107	11	0	0
0	0	0	0	0	0	0	0	0	0	0	98	255	255	197	255	255	255	255	255	255	248	115	0	0
0	0	0	0	0	0	4	62	116	184	255	255	255	255	255	255	214	166	107	47	11	0	0	0	0
0	0	0	104	184	255	255	255	255	255	255	255	255	243	43	0	0	0	0	0	0	0	0	0	0
0	0	0	0	40	73	73	73	47	98	255	255	255	248	54	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	4	178	255	255	255	255	243	22	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	60	255	255	255	255	255	255	174	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	4	178	255	248	204	255	243	165	255	128	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	98	255	255	128	98	255	243	30	225	255	128	0	0	0	0	0	0	0
0	0	0	0	0	0	34	238	255	205	11	98	255	243	22	60	255	255	128	0	0	0	0	0	0
0	0	0	0	0	8	225	255	248	54	0	98	255	243	22	0	141	255	255	128	0	0	0	0	0
0	0	0	0	4	178	255	255	89	0	0	141	255	243	22	0	8	225	255	255	128	0	0	0	0
0	0	0	0	141	255	255	128	0	0	0	141	255	243	22	0	0	60	255	255	255	157	11	0	0
0	0	0	98	255	255	128	0	0	0	0	141	255	243	22	0	0	0	141	255	255	255	209	54	0
0	0	98	255	255	128	0	0	0	0	4	178	255	243	22	0	0	0	8	225	255	255	255	255	157
0	98	255	248	54	0	0	0	0	0	8	225	255	248	54	0	0	0	0	60	255	255	255	255	255
134	248	157	11	0	0	0	0	0	0	34	238	255	248	54	0	0	0	0	0	40	140	197	197	197
69	11	0	0	0	0	0	0	0	0	34	238	255	243	22	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	60	255	255	243	22	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	34	238	255	205	11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 1.3 Digital image in matrix form (gray-level)

After vision discretization, a matrix of data that encode the captured physical image of world entities is obtained from the matrix of perceptive cells. For each perceptive cell, its position in the matrix reflects its corresponding world entity point's direction with respect to the camera (namely projective position), and its associated matrix value reflects its corresponding world entity point's visual brightness. Such matrix of data is what is referred to as a **digital image**, from which its corresponding original physical image of world entities can be decoded. An example of digital image that depicts a Chinese character "wood" is illustrated in Fig. 1.3.

Each element in the matrix of perceptive cells or in the digital image is called a **pixel**, and its associated discrete value is called its **pixel value**. A world entity point's corresponding pixel position in the digital image depends on the way of space discretization, whereas its corresponding pixel value depends on the way of value discretization.

Space discretization is rather flexible and can be realized in many ways. For example, digital cameras may generate digital images of size 320 pixel by 240 pixel, 640 pixel by 480 pixel, 1920 pixel by 1080 pixel, etc. In contrast, value discretization is rather rigid and is usually realized in few conventional ways. A conventional way of value discretization is to assign pixel values in the range of 0 to 255 which is right the range of integers representable by a byte. In such conventional way, the pixel value of 0 represents no brightness at all (namely black), whereas the pixel value of 255 represents the highest brightness (namely white)—Such conventional way of value discretization is not only applicable to gray-level digital images (such as illustrated in Fig. 1.3), but also applicable to color digital images after adaptation. For a color digital image, value discretization is performed for three channels namely the red-green-blue or RGB channels, and the range of 0 to 255 covers all discrete degrees of light intensity of the red, green, blue color channels respectively.

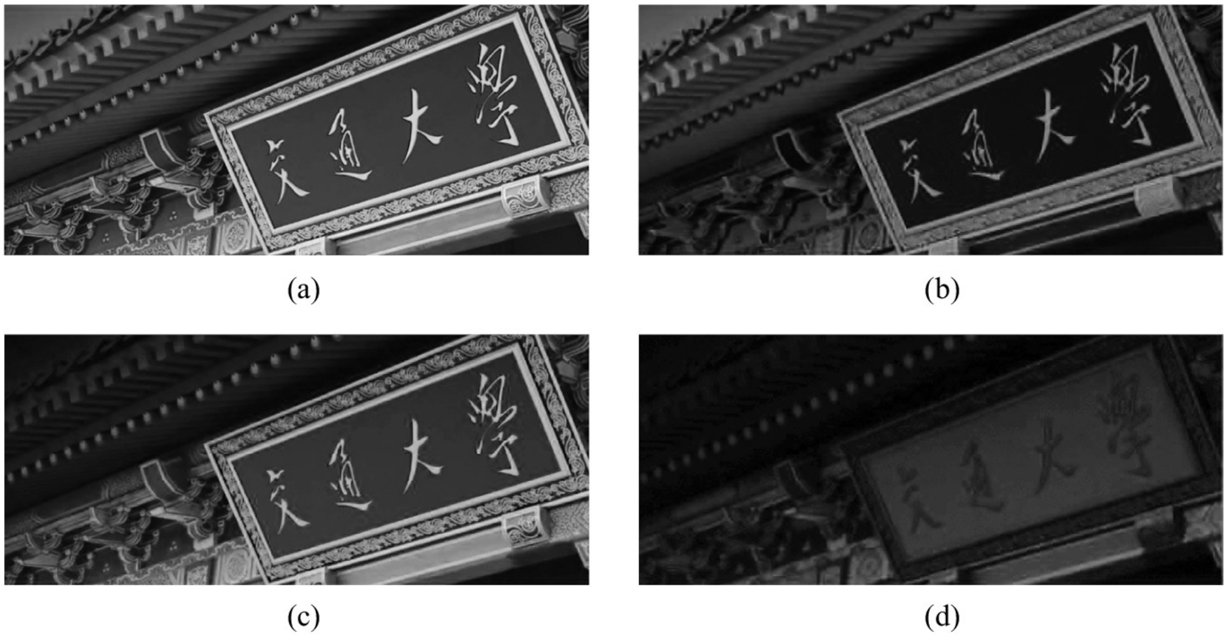
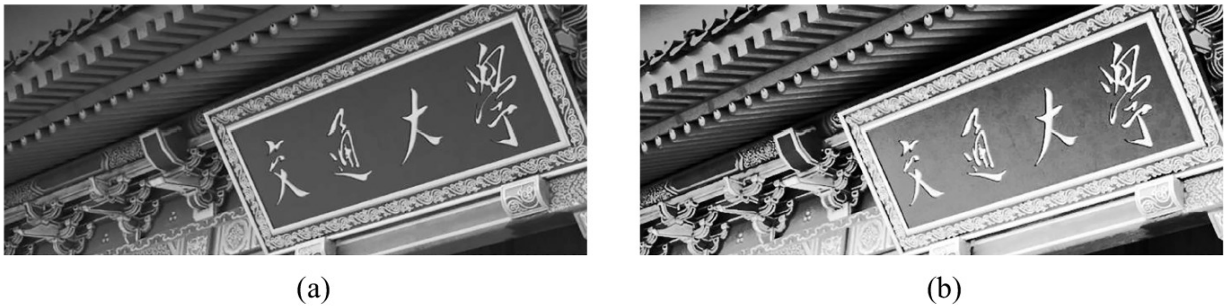


Fig. 1.4 Split red-green-blue channel images, **a** original color image; **b** red channel image; **c** green channel image; **d** blue channel image

An example of color digital image (or simply color image) is illustrated in Fig. 1.4a. The red channel image with value discretization in the range of 0–255 is illustrated in Fig. 1.4b. For the red channel, the pixel value of 0 represents no light intensity of red at all, whereas the pixel value of 255 represents the highest light intensity of red. The green and blue channel images with value discretization in the range of 0–255 are illustrated respectively in Fig. 1.4c, d. As for the red channel, for the green or blue channel, the pixel value of 0 represents no light intensity of green or blue at all, whereas the pixel value of 255 represents the highest light intensity of green or blue.

As mentioned previously, digital devices including digital cameras are ubiquitous nowadays. So throughout this book, what the author means by a camera and an image are by default a digital camera and a digital image [2]—It is worth noting that the concept of image for computer vision is not limited to images captured by normal cameras, but also includes images captured by special cameras, such as infra-red images, magnetic images, and event-triggered images.

The domain of image processing is important to the kingdom of computer vision. It includes many interesting and useful areas such as image enhancement, image filtering, image encoding-decoding, image restoration, image reconstruction, and image compression. An example of image enhancement via histogram specification (or histogram matching) is illustrated in Fig. 1.5. Figure 1.5a gives the original gray-level image before image enhancement, whereas Fig. 1.5b gives the enhanced image obtained from the original gray-level image via histogram specification.



(a) (b)

Fig. 1.5 Image enhancement, **a** original gray-level image; **b** enhanced image obtained via histogram specification

Although image processing plays an important role for computer vision, the author will refrain from deviating too much into presentation of image processing topics that are not related or at least not directly related to two-dimensional geometry analysis of images, but will present such knowledge only for basic necessity. To focus on geometry fundamentals of computer vision, the author will present image processing methods that are intended for extracting geometric entities such as points, edges, and corners from images and for related two-dimensional geometry analysis.

For example, given a Chinese character image illustrated in Fig. 1.6a, the target object in the image is the pair of Chinese characters “LI HAO”.[3] Character edges, which are geometric entities associated with the target object, are extracted and illustrated in Fig. 1.6b. Based on geometric information of extracted Chinese character edges, one may further compute the geometric center of these edges, as illustrated in Fig. 1.6c. Then the method of extracting such Chinese character edges (geometric entities) belongs to image processing knowledge that the author will present in this book.

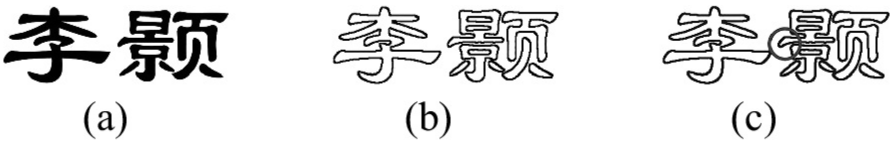
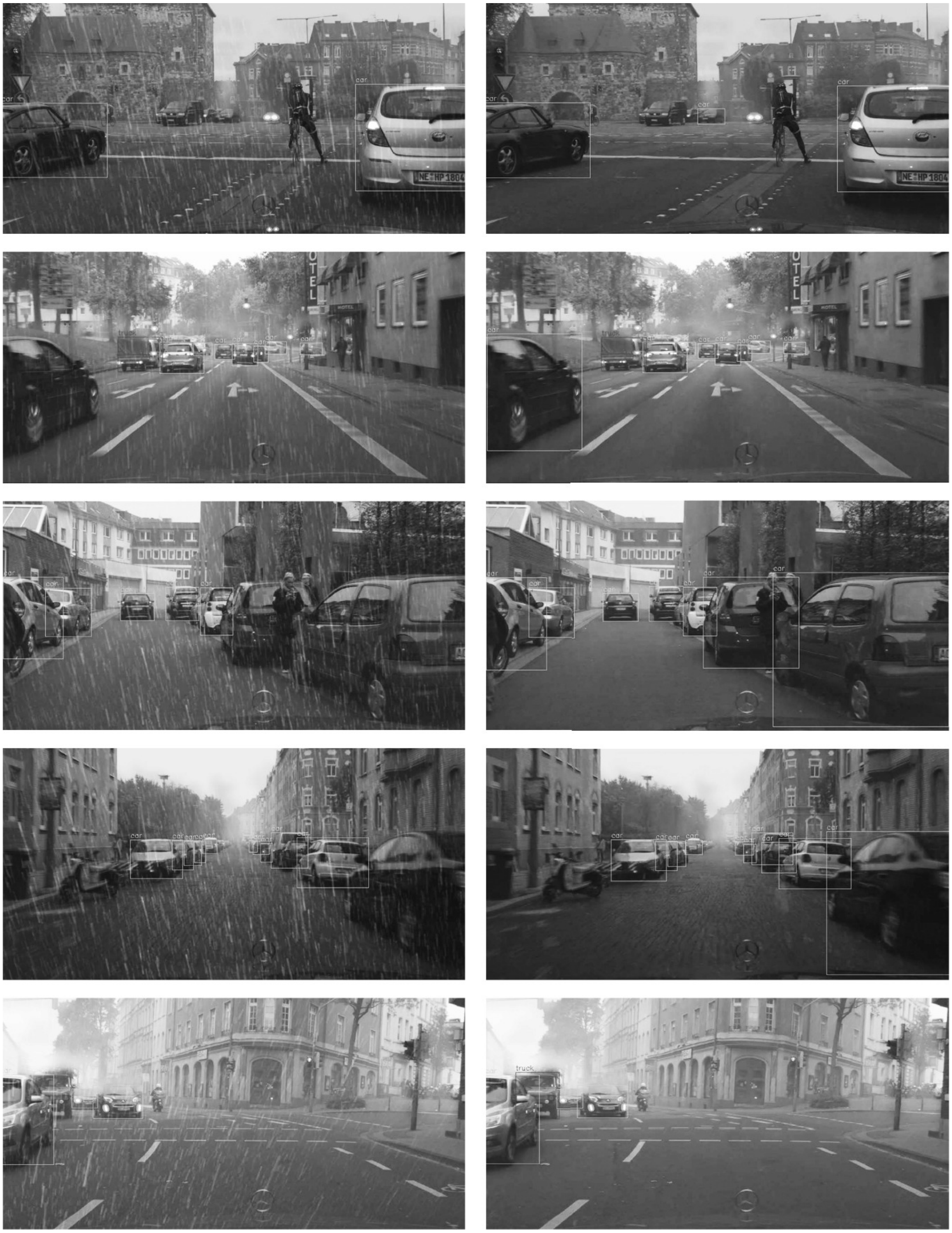


Fig. 1.6 Character edges, **a** Chinese character image; **b** Chinese character edges; **c** geometric center of these edges



(a) (b)

Fig. 1.7 Deraining **a** original rainy images (from the RainCityscapes dataset [3]); **b** deraining effect obtained via pattern recognition

The domain of pattern recognition is also important to the kingdom of computer vision and includes many fascinating topics such as object recognition, classification, feature selection, template matching, clustering. An example of deraining [4] (namely removing rain from images) which is useful in outdoor applications is illustrated in Fig. 1.7. Deraining scenarios might seem difficult to readers, yet there are even more challenging pattern recognition scenarios such as that illustrated in Fig. 1.8.

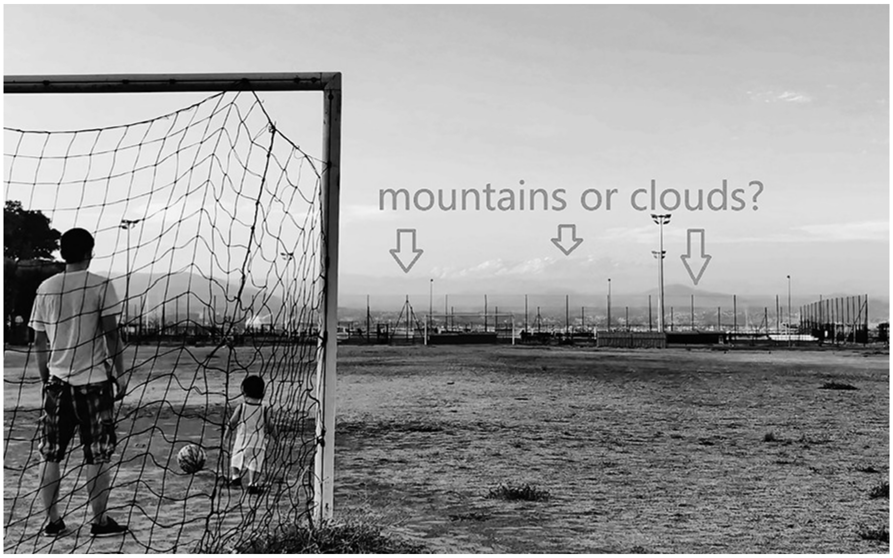


Fig. 1.8 Difficult pattern recognition: mountains or clouds?[4]

Although pattern recognition also plays an important role for computer vision, the author will refrain from deviating too much into presentation of pattern recognition topics that are not related to two-dimensional geometry analysis of images, but will present such knowledge only for basic necessity. To focus on geometry fundamentals of computer vision, the author will present pattern recognition methods that are intended for obtaining and organizing geometric entities such as shapes and structures in images and for related two-dimensional geometry analysis.

For example, given a round traffic sign, purely from the viewpoint of geometry, one does not care the property that it is a traffic sign (recognition of it as a traffic sign is realized by certain pattern recognition method), but only treats it as a circle and cares its geometric properties (position, size, etc.) in the image and in the world.

It is worth noting that sometimes to obtain desired geometric entities in two-dimensional images, one may consider their various kinds of properties as abstract points in certain high dimensional feature space. However, even for such kinds of cases, the author still focuses on handling the geometric entities' properties geometrically in the high dimensional feature space, and what are finally cared are their two-dimensional geometric properties (namely the two-dimensional part of their entire properties) in images. After all, it is the geometric entities' two-dimensional geometric properties in images that determine how they can be associated geometrically with their three-dimensional counterparts in the real world.

1.3 Computer Vision and Three-Dimensional Geometry Analysis

Not limited to two-dimensional geometry analysis within images, charms of computer vision also consist in its power for three-dimensional geometry analysis of the real world. In fact, explicit conception and usage of the geometric relationship between the three-dimensional world and the two-dimensional view can even date back to the era of renaissance, such as in realistic painting as illustrated in Fig. 1.9.

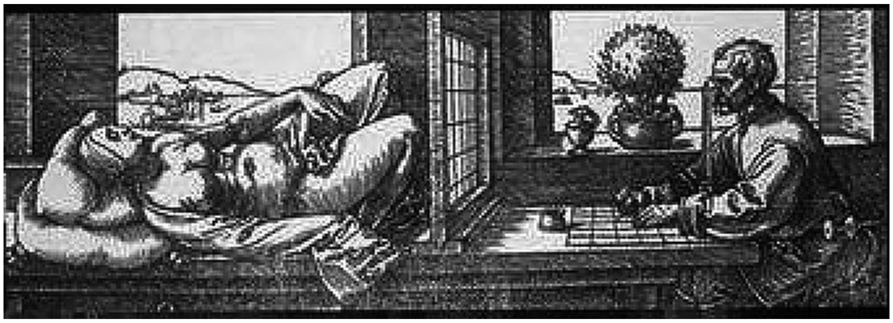


Fig. 1.9 Perspective painting via visual mapping [5]

To further facilitate understanding of such three-dimensional geometry analysis, readers may think of a number of practical applications that involve three-dimensional geometry analysis of the real world and reflect on how computer vision plays a core role in them:

- *Virtual file scanning*: In daily life, suppose one would like to scan a paper. Unfortunately, she or he does not have any scanner at hand, but instead has only a camera at hand. In such case, can she or he still generate a scanned image of the paper? The answer is indeed *yes*. For the purpose, one can use the camera to take a photo of the paper, extract the four corners of the paper, and do some further processing to generate a normal view image of the paper, as if she or he obtained an intended scanned image. An example effect of virtual file scanning is illustrated in Fig. 1.10a. How is this realized?
- *Image distortion rectification*: Wide-angle cameras are sometimes adopted in practical applications. An image captured by a wide-angle camera usually suffers from distortion. Intuitive interpretation of such distortion is that lines or line patterns in the image are bent and contracted towards the image center. However, it is not always desirable to use the raw distorted image directly in practical applications, because many useful aspects of geometry invariance are nullified by image distortion and nonlinear formalisms that model image distortion incur inconvenience in terms of mathematical analysis. Can one

process the raw distorted image to generate a rectified image which is exempt from image distortion? If so, as illustrated in Fig. 1.10b. How is this realized?

- *Visual SLAM* namely *visual simultaneous localization and mapping*: The objective of visual SLAM is to reveal three-dimensional structures of the environment from multiple images (views) of the environment. To reveal three-dimensional structures of the environment from two images, one needs to handle the core problem of how to estimate the inter-camera relative pose (i.e. the relative pose between the two camera poses) from the pair of images. An example effect of visual SLAM is illustrated in Fig. 1.10c. How is this realized?
- *Image stitching*: Cameras that allow image capturing in the panorama mode are commonly available nowadays and readers may have used the camera panorama function. Such function is based on image stitching, which refers to the practice of stitching multiple images that capture various views of the environment into a larger merged image that gives a panorama of the environment. An example effect of image stitching is illustrated in Fig. 1.10d. How is this realized?

Besides three-dimensional geometry analysis of the real world, readers may also think of a number of practical applications that involve three-dimensional geometry analysis of certain virtual world or of certain virtual entities imaginarily augmenting the real world, and reflect on how computer vision plays a core role in them:

- *Virtual reality*: The author believes that the concept of virtual reality is not strange to readers, because virtual reality has already found its way in daily life. Perhaps the most well-known form of virtual reality in daily life is that involved in video games, for which realistic style virtual environments need to be simulated. A simple example effect of virtual reality, namely generating different views around a tetrahedron located on a square carpet, is illustrated in Fig. 1.11a. How is this realized?

- *Augmented reality*: Readers might be less familiar with augmented reality than virtual reality. It is worth distinguishing between the two “realities”. Virtual reality refers to generating completely synthetic images that simulate the real world as well as purely imagined reality, whereas augmented reality refers to augmenting images actually captured from the real world by virtual elements to facilitate conception of the images. An example augmented reality effect of seeing through a front vehicle is illustrated in Fig. 1.11b.

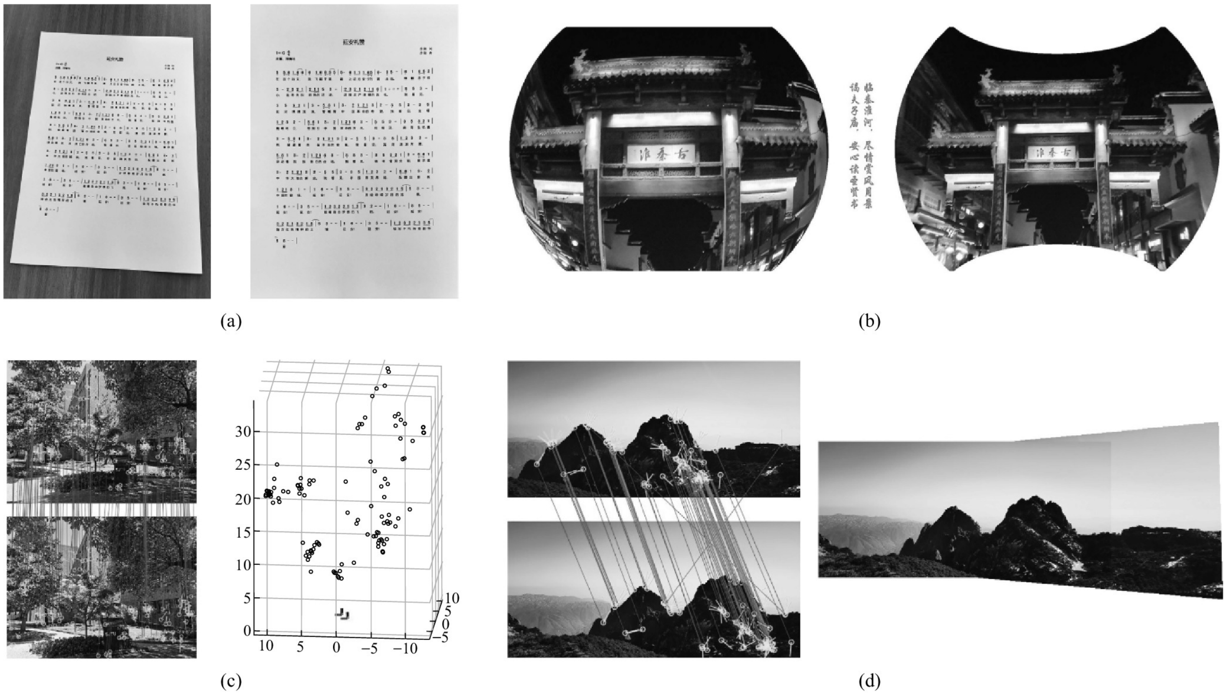


Fig. 1.10 Practical applications, **a** virtual file scanning; **b** image distortion rectification; **c** visual SLAM; **d** image stitching

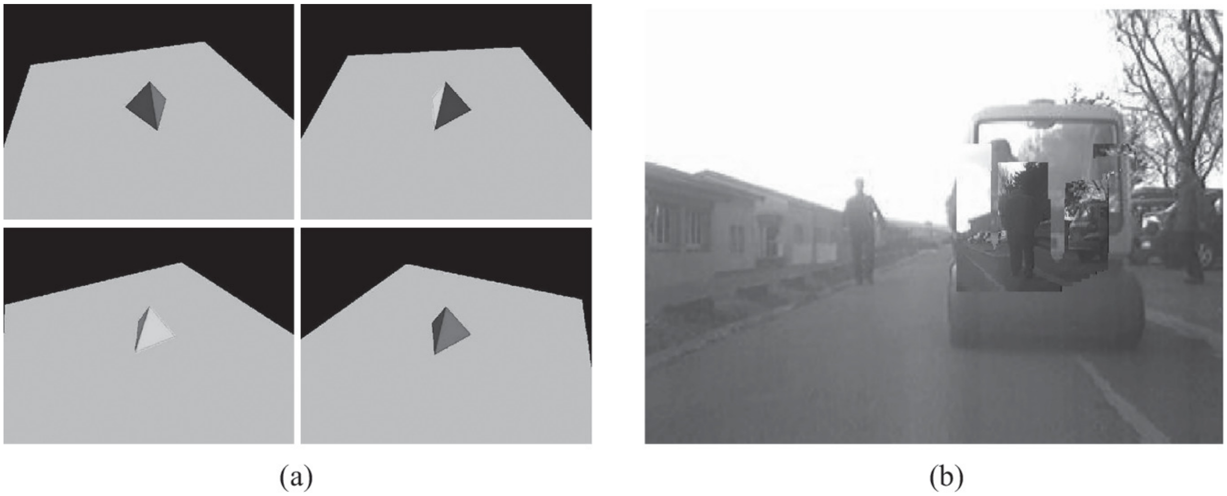


Fig. 1.11 Practical applications, **a** virtual reality of a tetrahedron; **b** augmented reality of seeing through [6]

Besides practical applications that involve three-dimensional geometry analysis of the real world as well as certain virtual world, various visual phenomena in daily life, some of which are taken so for granted as common sense, may also arouse people's interests, especially in reflecting on why they hold. For example, suppose image distortion of a camera is negligible, then real world lines in the image captured by the camera are still lines, whereas real world circles in the image captured by the camera become ellipses. The reasons behind such kinds of visual phenomena still involve three-dimensional geometry analysis rooted in the real world.

The author will present geometry fundamentals of computer vision that underlie three-dimensional geometry analysis involved in above kinds of practical applications as well as theoretical reflections.

1.4 Scope Preview

Focusing on *geometry fundamentals of computer vision*, this book is organized as follows.

Chapter 2 first defines the image coordinates system and clarifies the practice of treating an image as an image pixel value function, including image linear and quadratic interpolation as well as interpolation based image resolution enhancement. Chapter 2 then presents knowledge on extraction of intuitive feature points such as edge points and corner points. Chapter 2 also presents knowledge on extraction of abstract feature points such as the scale invariant feature transform (SIFT), especially clarifying valuable theory such as scale-space theory and ideas such as the Gaussian pyramid that underlie the SIFT and are generally applicable. Finally, Chapter 2 briefly introduces fundamentals of machine learning based feature extraction.

Chapter 3 presents knowledge on how to theoretically analyse geometric entities and how to effectively represent and compute their geometric properties in practice. Such kind of knowledge, known as computational geometry, aims at bridging the gap between what one can envisage and analyse for geometric entities and what one can realize (compute) for them. Chapter 3 describes how to compute a variety of distances and measures, how to represent two-dimensional shapes as graphs especially as planar straight-line graphs and perform relevant geometry analysis with or without convenience brought by convex geometry. Finally, Chap. 3 presents knowledge on how to handle closest point problems which are rather commonly encountered in practical applications.

Chapters 2 and 3 concern two-dimensional geometry analysis of images, whereas Chaps. 4 and 5 concern three-dimensional geometry analysis of images. Chapter 4 first presents fundamentals of projective geometry and demonstrates its charms with a number of interesting geometry examples. Chapter 4 then introduces camera models including the pinhole camera model (also known as the distortion-free camera model) and the distortion-considered camera model , which play an indispensable role for computer vision in a wide range of practical applications. Chapter 4 also introduces homography which is a special and important case of the spatial relationship (i.e. perspective mapping) described by the pinhole camera model . Finally, Chap. 4 presents geometry fundamentals of stereo vision, including the epipolar constraint, stereo vision rectification, and binocular disparity.

Chapter 5 complements Chap. 4 by providing methods of effective camera calibration that enable knowledge presented in Chap. 4 to actually take effect in practical applications. More specifically, Chap. 5 presents methods of calibrating the camera intrinsic parameters that are needed in a wide range of practical applications. Chapter 5 further presents knowledge on image distortion calibration that is also useful, especially when wide-angle cameras are adopted. Besides, Chap. 5 presents knowledge on stereo camera calibration and camera-range sensor calibration.

Chapter 6 concerns both two-dimensional geometry analysis and three-dimensional geometry analysis of images. Chapter 6 presents pattern recognition methods that are intended for obtaining and organizing two-dimensional and three-dimensional geometric entities in and among images. For the purpose, Chap. 6 first introduces representative clustering methods, then describes how to realize known feature-model relationship oriented model fitting namely to organize features points by fitting them with certain parametric model where the feature-model relationship is known, and finally describes how to realize unknown feature-model relationship oriented model fitting namely to organize features points by fitting them with certain parametric model where the feature-model relationship is unknown.

Chapter 7 presents a variety of interesting and useful practical applications in which geometry fundamentals of computer vision play an important role. The demonstrated practical applications include ground object detection, virtual reality, augmented reality (and cooperative augmented reality), image stitching, special tag based visual localization, and visual SLAM namely visual simultaneous localization and mapping. They necessitate comprehensive and flexible utilization of geometry fundamentals of computer vision.

1.5 Code Demonstration for Practice

The author believes that complete code demonstration is especially important and even indispensable to strengthening readers’ understanding of knowledge presented in this book and to enhancing their ability to flexibly apply relevant knowledge in practice, though some Matlab built-in functions are indeed available. The reason is that for geometry fundamentals of computer vision, the gap between how humans comprehend relevant knowledge and how computers can effectively put corresponding logic into practice does exist inevitably. Complete code demonstration aims right at bridging such gap, clarifying algorithmic implementations of various computer vision methods for practical applications.

It is worth noting a number of points concerning demonstrated algorithmicimplementations. On one hand, the author cannot scrutinize all aspects of “cleverness” in algorithmic implementations. Reasons are two-folds: First, the author can by no means have expertise in all aspects of algorithmic implementations. Even for any single aspect of involved algorithmic implementations, there would always be more competitive implementation methods that have been reported. Second, it is unnecessary to do so, because many aspects of “cleverness” in algorithmic implementations are not related to core knowledge on geometry fundamentals of computer vision and may even distract readers’ attention from the presented core knowledge. In one word, the demonstration code had better be of *conciseness* to highlight the core knowledge on geometry fundamentals of computer vision.

On the other hand, the author cannot be satisfied with “naive” code implementation simply for sake of demonstration completeness. After all, all pieces of demonstration code are intended for facilitating readers’ understanding of abstract knowledge as well as concrete application of relevant knowledge, instead of serving purely for “demonstration”. The author hopes that all pieces of demonstration code would enable readers to know how to flexibly adapt them into implementation in other programming languages and how to actually apply the presented knowledge in practical applications, probably not in the “cleverest” (most efficient, most desirable etc.) way but at least in a “clever” (efficient, desirable etc.) enough way. In one word, the author hopes that the provided demonstration code would also be beneficial to readers in terms of *practicalness*.

Above two concerns, namely the *conciseness* concern and the *practicalness* concern, tend to somehow contradict each other, and the author should treat both concerns dialectically for demonstration code throughout this book. Yet the author has to admit that such dialectic treatment is easier to say than to do. To guide tentative efforts for such dialectic treatment, the author roughly follows the principles below.

First, the author saves data properness checking and exception handling in demonstration code. For example, whether the dimensions of matrices under operation are mutually consistent? If not so, how to handle the exception of data dimension inconsistency? For another example, what if given data incur numeric singularity or are insufficient for qualified processing? The author saves all such kinds of consideration in demonstration code and take it for granted that all data involved in execution of demonstration code are proper and no exception handling is needed. However, readers had better bear in mind that data properness checking and exception handling are indispensable in practical applications.

Second, the author saves advanced algorithmic implementations of too fundamental operations in demonstration code, though the author may sometimes briefly mention the underlying knowledge behind. For example, consider the fundamental operation of matrix multiplication, the routine procedure of which is to perform element-wise multiplication of the *r*-th row of the first matrix and the *c*-th column of the second matrix and then add the multiplication results together to form the *r*-th row and *c*-th column element of the new matrix. Technique men perhaps take it for granted that matrix multiplication should be done using this so natural routine procedure. The routine procedure is of computational complexity $O(N^3)$, whereas faster procedures of matrix multiplication, such as Strassen's method of computational complexity $O(N^{\log 7}) \approx O(N^{2.807})$ [7] and Coppersmith-Winograd's method of computational complexity $O(N^{2.376})$ [8], do exist. Even for the more familiar example of multi-digit number multiplication (viewed as a special case of polynomial multiplication), the routine procedure of which is of computational complexity $O(N^2)$, faster procedures do exist as well [9, 10]. Despite beauty of such kinds of more efficient fundamental operations,[5] the author saves them in demonstration code but just uses Matlab built-in functions "$+$" or "$-$" (matrix addition or subtraction), "$*$" (matrix multiplication), "inv" (matrix inversion), "$\backslash$", i.e. "matrix left division" (linear equation solving), "eig" (eigenvalues and eigenvectors solving) and so on for involved numeric operations—In fact, many advanced algorithmic implementations have already been incorporated into Matlab for built-in functions.

Third, the author saves *ad hoc* implementation considerations and refinements that are not essential or general to core knowledge on geometry fundamentals of computer vision, especially those dedicated to special cases of practical applications or to special operation conditions. For example, the author refrains from demonstrating implementations that involve only integers or implementations that take advantage of parallel processing, though such kinds of implementations do have their positions in practical applications.

Fourth, the author tends to incline towards the *practicalness* concern if the *conciseness* concern is not sacrificed too much[6] and if above three principles are followed. Whenever there are crucial implementation points that might not be easily conceivable or comprehensible to readers, the author tends to not only provide demonstration code but also give relevant explanations.

It is worth noting again that it is difficult to figure out a perfect dialectic treatment of the *conciseness* concern and the *practicalness* concern for demonstration code and above principles are just followed roughly (but not strictly) to guide tentative efforts for such dialectic treatment of both concerns—If some readers come to the author’s office, argue that some pieces of demonstration code are inconsistent with the stated principles, and even argue that the stated principles are themselves not perfect but need improvements, the author will be totally open-minded to criticisms, agree with them, and be on their side—After all, concerning demonstration code throughout this book, the author’s objective is neither to advocate above principles themselves nor to verify consistency between the presented demonstration code and the principles. Instead, the author’s ultimate objective is that the provided demonstration code would indeed be of help to readers, especially in terms of strengthening their understanding of relevant knowledge and enhancing their ability to flexibly apply relevant knowledge in practice. The author believes that the demonstration code would achieve this ultimate objective.

References

1. R. Gonzalez, R. Woods. *Digital Image Processing*. Pearson Education (2010)
2. S. Theodoridis, K. Koutroumbas. *Pattern Recognition*. Elsevier (2006)
3. X. Hu, C. Fu, L. Zhu, and P. Heng. Depth-attentional features for single-image rain removal, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp 8022–8031 (2019)
4. C. Chen, H. Li. Robust representation learning with feedback for single image deraining, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7742–7751 (2021)
5. O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press (1993)
6. H. Li. *Cooperative Perception: Application In The Context Of Outdoor Intelligent Vehicle Systems*. Ph.D. Thesis, Mines ParisTech (2012)
7. V. Strassen, Gaussian elimination is not optimal. Numerische Mathematik **13**(4), 354–356 (1969) [MathSciNet][Crossref][zbMATH]
8. D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990) [MathSciNet][Crossref][zbMATH]
9. P. Burgisser, M. Clausen, A. Shokrollahi. *Algebraic Complexity Theory*. Springer (1997)
10. J. Gathen, J. Gerhard. *Modern Computer Algebra*. Cambridge University Press (1999)

Footnotes

[1] *Space discretization* is also known as *sampling*, whereas *value discretization* is also known as *quantization*.

[2] When people talk about images in the context of computer vision, they must mean a digital image because computers are digital devices that can only handle digital images. Similarly, when people talk about image processing nowadays, they also tend to mean digital image processing by default. On the other hand, it is worth noting that analog cameras appeared earlier than digital cameras. Before appearance of digital devices, there was once an era when "image processing" could refer to processing analog images (such as those for analog televisions) by analog electronic circuits. Even earlier than appearance of analog electronic devices, "image processing" might refer to chemical processing of photo films captured by cameras in those old days.

[3] The first Chinese character "LI" that represents the author's family name means "plum (tree)" and also means "reason" and "principle" in terms of Chinese family name etymology, whereas the second Chinese character "Hao" that represents the author's given name means "white" and "bright".

[4] X. Tan asked his little boy: "Can you distinguish between mountains and clouds in the distance"? In fact, this would be a difficult pattern recognition problem not only for children but also for computers.

[5] In given examples of matrix multiplication and multi-digit number multiplication, realization of faster multiplication usually relies on elegant "paraphrasing" of multiplication in the spirit of *divide-and-conquer*. Say to compute

$$3141592653589793 \times 2718281828459046,$$

"paraphrase" it as

$$\begin{aligned} & (31415926 \cdot 10^8 + 53589793) \times (27182818 \cdot 10^8 + 28459046) \\ = \; & (31415926 \times 27182818) \cdot 10^{16} + 53589793 \times 28459046 + \\ & [(31415926 + 53589793) \times (27182818 + 28459046) - \\ & 31415926 \times 27182818 - 53589793 \times 28459046] \cdot 10^8, \end{aligned}$$

which reduces the multiplication cost from that of the original large 16-digit-by-16-digit multiplication to that of three smaller 8-digit-by-8-digit multiplications, with moderate increase in the addition cost. Note that multiplication normally costs much more than addition, such “paraphrasing” tends to decrease the overall cost and can be applied recursively to further decrease the computation cost. The computational complexity $C(N)$ for such multi-digit number multiplication follows roughly the recursive relationship

$$C(N) = 3\,C(N/2) + O(N)$$

which gives $C(N) = O(N^{\log 3}) \approx O(N^{1.585})$, considerably smaller than $O(N^2)$.

[6] The author tends to incline towards the conciseness concern if it would otherwise be indeed sacrificed too much whereas there is no gain at the complexity order level. See “code demonstration note” in Appendix A.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_2

2. Image Processing for Feature Extraction

Hao Li[1]
(1) Shanghai Jiao Tong University, Shanghai, China

2.1 Image Coordinates System
2.1.1 From Discrete Indices to Continuous Coordinates
2.1.2 Linear Interpolation
2.1.3 Quadratic Interpolation
2.1.4 Image Resolution Enhancement
2.2 Feature Points
2.2.1 Thresholding: Pixel-Wise Point Extraction
2.2.2 Application of Image Thresholding: Watermark Removal
2.2.3 Adaptive Thresholding
2.2.4 Non-Maximum Suppression: Extremum Point Extraction
2.2.5 Dilation and Erosion: Contour Extraction
2.3 Edge Points
2.3.1 Gradient Image
2.3.2 Directional Non-maximum Suppression
2.3.3 Double Thresholding
2.3.4 Canny Edge Detection
2.4 Corner Points
2.5 Scale-Space Feature Extraction
2.5.1 Scale Space: Filtered Images Associated with Various Filtering-Scale Parameter
2.5.2 Gaussian Pyramid
2.5.3 Scale-Invariant Laplacian
2.5.4 Difference-of-Gaussian
2.6 Scale Invariant Feature Transform (SIFT)
2.6.1 Feature Point Localization
2.6.2 Local Image Orientation Assignment
2.6.3 Local Image Descriptor Generation
2.7 Post-SIFT Features
2.7.1 Fast Filtering: Example of SURF
2.7.2 Fast Feature Detection: Example of FAST
2.7.3 Simplified Feature Description: Example of BRIEF
2.7.4 Oriented FAST and Rotated BRIEF (ORB)
2.8 Machine Learning Based Features
2.8.1 Pre-Deep Learning Methods
2.8.2 Deep Learning Methods
2.8.3 Learn for Feature Extraction
2.9 Note
References

2.1 Image Coordinates System

2.1.1 From Discrete Indices to Continuous Coordinates

As introduced in Sect.1.2 in Chap. 1, an image is a matrix of data obtained via vision discretization by a camera. A matrix can be naturally related to a two-dimensional coordinates system, because people normally locate matrix elements or cells in a way of two-dimensional indexing. For example, people are used to say “the element of the forth column and of the third row” and “the cell of the tenth column and of the second row”, or for short “the forth-column-third-row element” and “the tenth-column-second-row cell” respectively.

This convention reflects well the idea of describing and analysing an image in a two-dimensional coordinates system, as illustrated in Fig. 2.1. A two-dimensional coordinates system is established, with the origin $[0,0]^{\mathrm{T}}$ located at an imaginary position corresponding to the zeroth (0-th) column and the zeroth row of the image. The horizontal axis (denoted as $\boldsymbol{U}$) of the two-dimensional coordinates system is along the row direction and indexes columns, whereas the vertical axis (denoted as $\boldsymbol{V}$) of the two-dimensional coordinates system is along the column direction and indexes rows. Then each pixel is associated with a unique point in the established two-dimensional coordinates system and corresponds to a two-dimensional vector that consists of its horizontal coordinates u and vertical coordinates v. For example, the pixel $\boldsymbol{P}$ illustrated in Fig. 2.1 corresponds to the two-dimensional vector $[4,3]^{\mathrm{T}}$, which means the pixel $\boldsymbol{P}$ is located in the forth column (“C4”) and the third row (“R3”) of the matrix. Such two-dimensional coordinates system is called the **image coordinates system**.

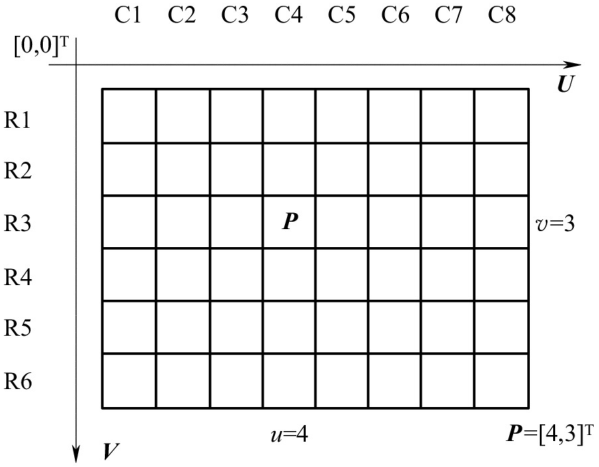


Fig. 2.1 Image coordinates system

Once the image coordinates system is established, the image’s pixel values can be treated as sampled values of an implicit continuous bivariate function defined on the image coordinates system. The implicit continuous bivariate function is called the **image pixel value function** and is denoted as $I(u, v)$. Treating an image as a set of samples of an image pixel value function $I(u, v)$ possesses the merit of enabling many mathematical tools such as real analysis and even complex analysis to be applicable to image processing.

Suppose the image height is H and the image width is W. Definition of the image pixel value function $I(u, v)$ on the positive integer bivariate domain (illustrated in Fig. 2.2)

$$\boldsymbol{I}_{\mathrm{D}} = \{[u,v]^{\mathrm{T}} | u \in \{1,2,\cdots,W\}, v \in \{1,2,\cdots,H\}\} \tag{2.1}$$

is obvious: For each $[u,v]^{\mathrm{T}} \in \boldsymbol{I}_{\mathrm{D}}$,

$$I(u,v) = \bar{I}(u,v), \tag{2.2}$$

where $\bar{I}$ denotes the matrix of pixel values that represents the image.

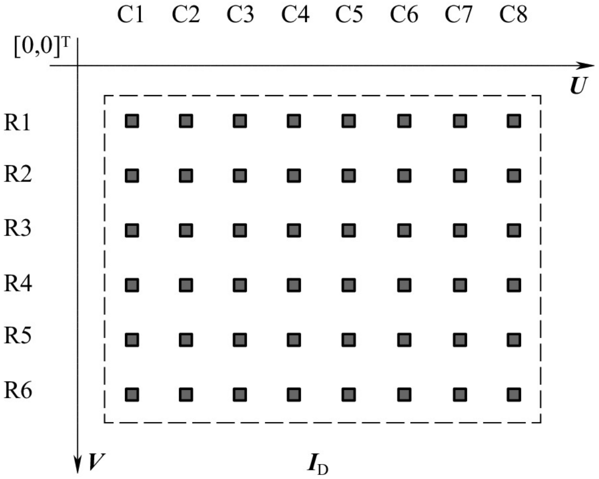


Fig. 2.2 Positive integer bivariate domain

On the other hand, the image pixel value function $I(u, v)$ is not defined a priori on any bivariate domain other than the positive integer bivariate domain $\boldsymbol{I}_{\mathrm{D}}$ in the image coordinates system. Since only the image area in the image coordinates system is normally concerned in practical applications, the author focuses on the real bivariate domain

$$\boldsymbol{I}_{\mathrm{C}} = \{[u,v]^{\mathrm{T}} | 1 \leqslant u \leqslant W, 1 \leqslant v \leqslant H\}, \tag{2.3}$$

which contains $\boldsymbol{I}_{\mathrm{D}}$ as subset (namely $\boldsymbol{I}_{\mathrm{D}} \subset \boldsymbol{I}_{\mathrm{C}}$).

Given a generic point $\boldsymbol{P} = [u,v]^{\mathrm{T}} \in \boldsymbol{I}_{\mathrm{C}}$ with non-integer coordinates, namely a generic point $\boldsymbol{P} \in \boldsymbol{I}_{\mathrm{C}}$ but $\boldsymbol{P} \notin \boldsymbol{I}_{\mathrm{D}}$. The image pixel value function I is not defined a priori at $\boldsymbol{P}$, yet one may estimate $I(\boldsymbol{P}) = I(u,v)$ by interpolating pixel values of its neighbours (neighbouring points) with integer coordinates. For example, one may estimate $I(\boldsymbol{P}) = I(u,v)$ via linear interpolation or quadratic interpolation.

2.1.2 Linear Interpolation

First, consider one-dimensional linear interpolation of pixel values of a point's neighbours. Given a generic point $\boldsymbol{P} = [u,v]^{\mathrm{T}}$, suppose v is an integer whereas u is a non-integer number. As illustrated in Fig. 2.3, let the image pixel value function $I(u, v)$ defined on the line segment between points $[\lfloor u \rfloor, v]^{\mathrm{T}}$ and $[\lfloor u \rfloor + 1, v]^{\mathrm{T}}$ be approximated by a linear univariate function [1]

$$I(u,v) = a_1 u + a_0 \tag{2.4}$$

or equivalently by

$$I(u,v) = (\lfloor u \rfloor + 1 - u)k_1 + (u - \lfloor u \rfloor)k_2, \tag{2.5}$$

which satisfies the pixel values at points $[\lfloor u \rfloor, v]^{\mathrm{T}}$ and $[\lfloor u \rfloor + 1, v]^{\mathrm{T}}$.

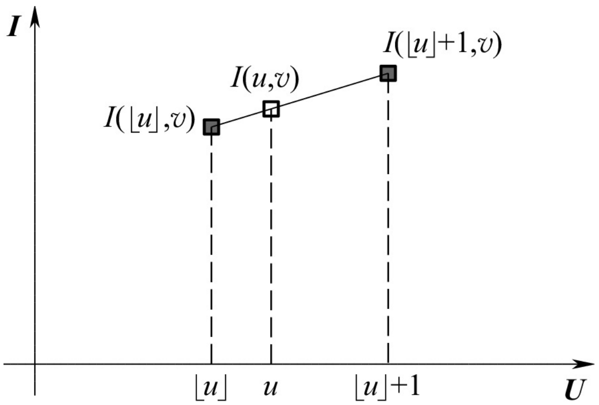


Fig. 2.3 One-dimensional linear interpolation of pixel values

Follow (2.5), solve

$$\begin{cases} (\lfloor u \rfloor + 1 - u)k_1 + (u - \lfloor u \rfloor)k_2|_{u=\lfloor u \rfloor} & = I(\lfloor u \rfloor, v) \\ (\lfloor u \rfloor + 1 - u)k_1 + (u - \lfloor u \rfloor)k_2|_{u=\lfloor u \rfloor+1} & = I(\lfloor u \rfloor + 1, v) \end{cases}$$

and obtain

$$k_1 = I(\lfloor u \rfloor, v), \quad k_2 = I(\lfloor u \rfloor + 1, v).$$

So based on the linear approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(\lfloor u \rfloor, v)$ and $I(\lfloor u \rfloor + 1, v)$, as

$$\begin{aligned} I(u,v) = & \ (\lfloor u \rfloor + 1 - u)k_1 + (u - \lfloor u \rfloor)k_2 \\ = & \ (\lfloor u \rfloor + 1 - u)I(\lfloor u \rfloor, v) + (u - \lfloor u \rfloor)I(\lfloor u \rfloor + 1, v) \\ = & \ (\left\lfloor u \right\rfloor + 1 - u)\bar{I}(\left\lfloor u \right\rfloor, v) + (u - \left\lfloor u \right\rfloor)\bar{I}(\left\lfloor u \right\rfloor + 1, v). \end{aligned} \tag{2.6}$$

Similarly, suppose u is an integer whereas v is a non-integer number. Based on the linear approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(u, \lfloor v \rfloor)$ and $I(u, \lfloor v \rfloor + 1)$, as

$$\begin{aligned} I(u,v) = & \ (\lfloor v \rfloor + 1 - v)I(u, \lfloor v \rfloor) + (v - \lfloor v \rfloor)I(u, \lfloor v \rfloor + 1) \\ = & \ (\left\lfloor v \right\rfloor + 1 - v)\bar{I}(u, \left\lfloor v \right\rfloor) + (v - \left\lfloor v \right\rfloor)\bar{I}(u, \left\lfloor v \right\rfloor + 1). \end{aligned} \tag{2.7}$$

Next, consider two-dimensional linear (bilinear) interpolation of pixel values of a point's neighbours. Given a generic point $\boldsymbol{P} = [u, v]^{\mathrm{T}}$; suppose both u and v are non-integer numbers. As illustrated in Fig. 2.4, let the image pixel value function $I(u, v)$ defined on the unit square area with corner points $[\lfloor u \rfloor, \lfloor v \rfloor]^{\mathrm{T}}$, $[\lfloor u \rfloor + 1, \lfloor v \rfloor]^{\mathrm{T}}$, $[\lfloor u \rfloor, \lfloor v \rfloor + 1]^{\mathrm{T}}$, and $[\lfloor u \rfloor + 1, \lfloor v \rfloor + 1]^{\mathrm{T}}$ be approximated by a bilinear function

$$I(u,v) = (a_1 u + a_0)(b_1 v + b_0) \tag{2.8}$$

or equivalently by

$$\begin{aligned} I(u,v) = & \ [(\lfloor u \rfloor + 1 - u)k_1 + (u - \lfloor u \rfloor)k_2][(\lfloor v \rfloor + 1 - v)k_3 + (v - \lfloor v \rfloor)k_4] \\ = & \ [(1 - \Delta u)k_1 + \Delta u k_2][(1 - \Delta v)k_3 + \Delta v k_4], \end{aligned} \tag{2.9}$$

which satisfies the pixel values at points $[\lfloor u\rfloor, \lfloor v\rfloor]^{\mathrm{T}}$, $[\lfloor u\rfloor+1, \lfloor v\rfloor]^{\mathrm{T}}$, $[\lfloor u\rfloor, \lfloor v\rfloor+1]^{\mathrm{T}}$, and $[\lfloor u\rfloor+1, \lfloor v\rfloor+1]^{\mathrm{T}}$. Here, $\Delta u \equiv u-\lfloor u\rfloor$, and $\Delta v \equiv v-\lfloor v\rfloor$.

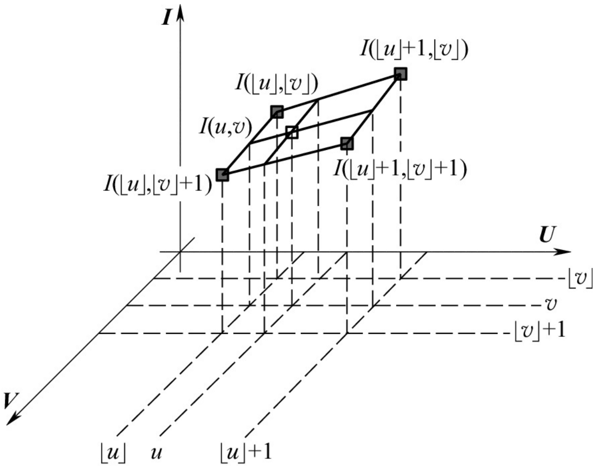


Fig. 2.4 Two-dimensional linear (bilinear) interpolation of pixel values

Follow (2.9), solve

$$\begin{cases} ((1-\Delta u)k_1+\Delta u k_2)((1-\Delta v)k_3+\Delta v k_4)|_{u=\lfloor u\rfloor, v=\lfloor v\rfloor} & = I(\lfloor u\rfloor, \lfloor v\rfloor) \\ ((1-\Delta u)k_1+\Delta u k_2)((1-\Delta v)k_3+\Delta v k_4)|_{u=\lfloor u\rfloor+1, v=\lfloor v\rfloor} & = I(\lfloor u\rfloor+1, \lfloor v\rfloor) \\ ((1-\Delta u)k_1+\Delta u k_2)((1-\Delta v)k_3+\Delta v k_4)|_{u=\lfloor u\rfloor, v=\lfloor v\rfloor+1} & = I(\lfloor u\rfloor, \lfloor v\rfloor+1) \\ ((1-\Delta u)k_1+\Delta u k_2)((1-\Delta v)k_3+\Delta v k_4)|_{u=\lfloor u\rfloor+1, v=\lfloor v\rfloor+1} & = I(\lfloor u\rfloor+1, \lfloor v\rfloor+1) \end{cases}$$

and obtain

$$\begin{aligned} k_1k_3 &= I(\lfloor u\rfloor, \lfloor v\rfloor), \\ k_2k_3 &= I(\lfloor u\rfloor+1, \lfloor v\rfloor), \\ k_1k_4 &= I(\lfloor u\rfloor, \lfloor v\rfloor+1), \\ k_2k_4 &= I(\lfloor u\rfloor+1, \lfloor v\rfloor+1). \end{aligned}$$

So based on the linear approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(\lfloor u\rfloor, \lfloor v\rfloor)$, $I(\lfloor u\rfloor+1, \lfloor v\rfloor)$, $I(\lfloor u\rfloor, \lfloor v\rfloor+1)$, and $I(\lfloor u\rfloor+1, \lfloor v\rfloor+1)$ as

$$\begin{aligned} I(u,v) = & [(1-\Delta u)k_1+\Delta u k_2][(1-\Delta v)k_3+\Delta v k_4] \\ = & (1-\Delta u)(1-\Delta v)k_1k_3+\Delta u(1-\Delta v)k_2k_3+(1-\Delta u)\Delta v k_1k_4+\Delta u\Delta v k_2k_4 \\ = & (1-\Delta u)(1-\Delta v)I(\lfloor u\rfloor, \lfloor v\rfloor)+\Delta u(1-\Delta v)I(\lfloor u\rfloor+1, \lfloor v\rfloor)+ \\ & (1-\Delta u)\Delta v I(\lfloor u\rfloor, \lfloor v\rfloor+1)+\Delta u\Delta v I(\lfloor u\rfloor+1, \lfloor v\rfloor+1) \\ = & (\lfloor u\rfloor+1-u)(\lfloor v\rfloor+1-v)I(\lfloor u\rfloor, \lfloor v\rfloor)+(u-\lfloor u\rfloor)(\lfloor v\rfloor+1-v) \\ & I(\lfloor u\rfloor+1, \lfloor v\rfloor)+(\lfloor u\rfloor+1-u)(v-\lfloor v\rfloor)I(\lfloor u\rfloor, \lfloor v\rfloor+1)+ \\ & (u-\lfloor u\rfloor)(v-\lfloor v\rfloor)I(\lfloor u\rfloor+1, \lfloor v\rfloor+1). \end{aligned} \tag{2.10}$$

It is worth noting that the bilinear function is not the same to the linear bivariate function. Figure 2.4 might give readers an illusion that the surface formed by the bilinear function $I(u, v)$ described in (2.8) (2.9), or (2.10) is a plane in the three-dimensional space, yet such surface is not necessarily a plane. In fact, the surface formed by a bilinear function rarely happens to be a plane in the three-dimensional space.

2.1.3 Quadratic Interpolation

First, consider one-dimensional quadratic interpolation of pixel values of a point's neighbours. Given a generic point $\boldsymbol{P} = [u, v]^{\mathrm{T}}$; suppose v is an integer whereas u is a non-integer number. As illustrated in Fig. 2.5, let the image pixel value function $I(u, v)$ defined on the line segment between points $[\bar{u} - 1, v]^{\mathrm{T}}$ and $[\bar{u} + 1, v]^{\mathrm{T}}$ be approximated by a quadratic univariate function [2]

$$I(u, v) = a_2 u^2 + a_1 u + a_0 \tag{2.11}$$

or equivalently by

$$I(u, v) = (u - \bar{u})^2 k_2 + (u - \bar{u}) k_1 + k_0, \tag{2.12}$$

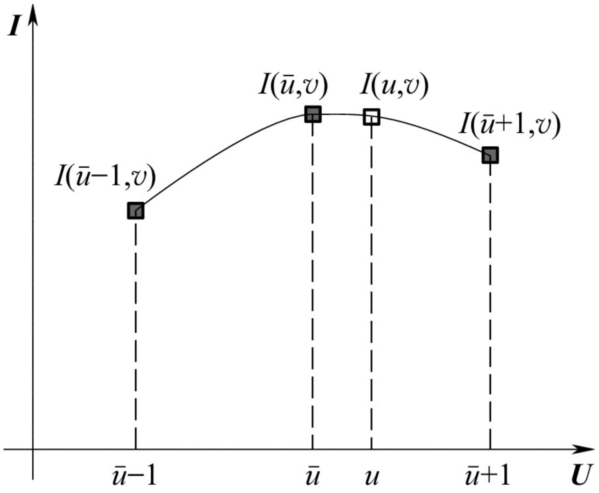


Fig. 2.5 One-dimensional quadratic interpolation of pixel values

which satisfies the pixel values at points $[\bar{u} - 1, v]^{\mathrm{T}}$, $[\bar{u}, v]^{\mathrm{T}}$ and $[\bar{u} + 1, v]^{\mathrm{T}}$. Follow (2.12), solve

$$\begin{cases} (u - \bar{u})^2 k_2 + (u - \bar{u}) k_1 + k_0|_{u=\bar{u}-1} & = I(\bar{u} - 1, v) \\ (u - \bar{u})^2 k_2 + (u - \bar{u}) k_1 + k_0|_{u=\bar{u}} & = I(\bar{u}, v) \\ (u - \bar{u})^2 k_2 + (u - \bar{u}) k_1 + k_0|_{u=\bar{u}+1} & = I(\bar{u} + 1, v) \end{cases}$$

namely

$$\begin{cases} k_2 - k_1 + k_0 & = I(\bar{u} - 1, v) \\ k_0 & = I(\bar{u}, v) \\ k_2 + k_1 + k_0 & = I(\bar{u} + 1, v) \end{cases}$$

and obtain

$$\begin{aligned} k_2 =& \ \tfrac{I(\bar{u}+1,v)+I(\bar{u}-1,v)-2I(\bar{u},v)}{2} \\ k_1 =& \ \tfrac{I(\bar{u}+1,v)-I(\bar{u}-1,v)}{2} \\ k_0 =& \ I(\bar{u},v). \end{aligned}$$

So based on the quadratic approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(\bar{u}-1,v)$, $I(\bar{u},v)$, and $I(\bar{u}+1,v)$ as

$$\begin{aligned} I(u,v) =& \ (u-\bar{u})^2 k_2 + (u-\bar{u})k_1 + k_0 \\ =& \ (u-\bar{u})^2 \tfrac{I(\bar{u}+1,v)+I(\bar{u}-1,v)-2I(\bar{u},v)}{2} \\ & +(u-\bar{u})\tfrac{I(\bar{u}+1,v)-I(\bar{u}-1,v)}{2} + I(\bar{u},v) \\ =& \ (u-\bar{u})^2 \tfrac{\bar{I}(\bar{u}+1,v)+\bar{I}(\bar{u}-1,v)-2\bar{I}(\bar{u},v)}{2} \\ & +(u-\bar{u})\tfrac{\bar{I}(\bar{u}+1,v)-\bar{I}(\bar{u}-1,v)}{2} + \bar{I}(\bar{u},v). \end{aligned} \tag{2.13}$$

Similarly, suppose u is an integer whereas v is a non-integer number. Based on the quadratic approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(u,\bar{v}-1)$, $I(u,\bar{v})$, and $I(u,\bar{v}+1)$ as

$$\begin{aligned} I(u,v) =& \ (v-\bar{v})^2 \tfrac{I(u,\bar{v}+1)+I(u,\bar{v}-1)-2I(u,\bar{v})}{2} \\ & +(v-\bar{v})\tfrac{I(u,\bar{v}+1)-I(u,\bar{v}-1)}{2} + I(u,\bar{v}) \\ =& \ (v-\bar{v})^2 \tfrac{\bar{I}(u,\bar{v}+1)+\bar{I}(u,\bar{v}-1)-2\bar{I}(u,\bar{v})}{2} \\ & +(v-\bar{v})\tfrac{\bar{I}(u,\bar{v}+1)-\bar{I}(u,\bar{v}-1)}{2} + \bar{I}(u,\bar{v}). \end{aligned} \tag{2.14}$$

Next, consider two-dimensional quadratic (biquadratic) interpolation of pixel values of a point's neighbours. Given a generic point $\boldsymbol{P} = [u,v]^{\mathrm{T}}$; suppose both u and v are non-integer numbers. Let the image pixel value function $I(u, v)$ defined on the unit square area with corner points $[\bar{u}-1,\bar{v}-1]^{\mathrm{T}}$, $[\bar{u}+1,\bar{v}-1]^{\mathrm{T}}$, $[\bar{u}+1,\bar{v}+1]^{\mathrm{T}}$, and $[\bar{u}-1,\bar{v}+1]^{\mathrm{T}}$ be approximated by a biquadratic function

$$I(u,v) = (a_2u^2 + a_1u + a_0)(b_2v^2 + b_1v + b_0) \tag{2.15}$$

or equivalently by

$$\begin{aligned} I(u,v) =& \ [(u-\bar{u})^2k_2 + (u-\bar{u})k_1 + k_0][(v-\bar{v})^2k_5 + (v-\bar{v})k_4 + k_3] \\ =& \ (\Delta u^2k_2 + \Delta uk_1 + k_0)(\Delta v^2k_5 + \Delta vk_4 + k_3) \\ =& \ \Delta u^2\Delta v^2k_2k_5 + \Delta u^2\Delta vk_2k_4 + \Delta u^2k_2k_3 \\ & +\Delta u\Delta v^2k_1k_5 + \Delta u\Delta vk_1k_4 + \Delta uk_1k_3 + \Delta v^2k_0k_5 + \Delta vk_0k_4 + k_0k_3 \end{aligned} \tag{2.16}$$

which satisfies the pixel values at nine points $[\bar{u}+i,\bar{v}+j]^{\mathrm{T}}$ $(i \in \{-1,0,+1\}$, $j \in \{-1,0,+1\})$. Here, $\Delta u \equiv u - \bar{u}$, and $\Delta v \equiv v - \bar{v}$.

Follow (2.16), consider

$$\begin{aligned}&\Delta u^2\Delta v^2 k_2k_5+\Delta u^2\Delta v k_2k_4+\Delta u^2 k_2k_3+\Delta u\Delta v^2 k_1k_5+\Delta u\Delta v k_1k_4\\&+\Delta u k_1k_3+\Delta v^2 k_0k_5+\Delta v k_0k_4+k_0k_3|_{u=\bar u+i,v=\bar v+j}=I(\bar u+i,\bar v+j)\end{aligned}$$

namely

$$\begin{cases}k_2k_5-k_2k_4+k_2k_3-k_1k_5+k_1k_4-k_1k_3+k_0k_5-k_0k_4+k_0k_3 & =I(\bar u-1,\bar v-1)\\ k_0k_5-k_0k_4+k_0k_3 & =I(\bar u,\bar v-1)\\ k_2k_5-k_2k_4+k_2k_3+k_1k_5-k_1k_4+k_1k_3+k_0k_5-k_0k_4+k_0k_3 & =I(\bar u+1,\bar v-1)\\ k_2k_3-k_1k_3+k_0k_3 & =I(\bar u-1,\bar v)\\ k_0k_3 & =I(\bar u,\bar v)\\ k_2k_3+k_1k_3+k_0k_3 & =I(\bar u+1,\bar v)\\ k_2k_5+k_2k_4+k_2k_3-k_1k_5-k_1k_4-k_1k_3+k_0k_5+k_0k_4+k_0k_3 & =I(\bar u-1,\bar v+1)\\ k_0k_5+k_0k_4+k_0k_3 & =I(\bar u,\bar v+1)\\ k_2k_5+k_2k_4+k_2k_3+k_1k_5+k_1k_4+k_1k_3+k_0k_5+k_0k_4+k_0k_3 & =I(\bar u+1,\bar v+1)\end{cases}$$

or

$$\boldsymbol{CK}\equiv\begin{bmatrix}1&-1&1&-1&1&-1&1&-1&1\\0&0&0&0&0&0&1&-1&1\\1&-1&1&1&-1&1&1&-1&1\\0&0&1&0&0&-1&0&0&1\\0&0&0&0&0&0&0&0&1\\0&0&1&0&0&1&0&0&1\\1&1&1&-1&-1&-1&1&1&1\\0&0&0&0&0&0&1&1&1\\1&1&1&1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}k_2k_5\\k_2k_4\\k_2k_3\\k_1k_5\\k_1k_4\\k_1k_3\\k_0k_5\\k_0k_4\\k_0k_3\end{bmatrix}=\begin{bmatrix}I(\bar u-1,\bar v-1)\\I(\bar u,\bar v-1)\\I(\bar u+1,\bar v-1)\\I(\bar u-1,\bar v)\\I(\bar u,\bar v)\\I(\bar u+1,\bar v)\\I(\bar u-1,\bar v+1)\\I(\bar u,\bar v+1)\\I(\bar u+1,\bar v+1)\end{bmatrix}\equiv\boldsymbol{I}_{\bar u,\bar v},\qquad(2.17)$$

where $\boldsymbol{C}$ denotes the 9-by-9 matrix on the left side of (2.17), $\boldsymbol{K}$ denotes the 9-by-1 vector on the left side of (2.17), and $\boldsymbol{I}_{\bar u,\bar v}$ denotes the 9-by-1 vector on the right side of (2.17).

Explanations hover over how $\boldsymbol{CK}$ in (2.17) is obtained. In fact, $\boldsymbol{CK}$ can be obtained conveniently via *matrix vectorization* [3] with the help of *Kronecker product* [1][4]:

$$\begin{aligned}\boldsymbol{CK}= & \quad\mathrm{vec}\left(\begin{bmatrix}(-1)^2\cdot k_2+(-1)\cdot k_1+k_0\\0^2\cdot k_2+0\cdot k_1+k_0\\(+1)^2\cdot k_2+(+1)\cdot k_1+k_0\end{bmatrix}\right.\\ & \quad\left.\begin{bmatrix}(-1)^2\cdot k_5+(-1)\cdot k_4+k_3 & 0^2\cdot k_5+0\cdot k_4+k_3 & (+1)^2\cdot k_5+(+1)\cdot k_4+k_3\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix}\right)\\ = & \quad\mathrm{vec}\left(\begin{bmatrix}1&-1&1\\0&0&1\\1&1&1\end{bmatrix}\begin{bmatrix}k_2\\k_1\\k_0\end{bmatrix}\begin{bmatrix}k_5&k_4&k_3\end{bmatrix}\begin{bmatrix}1&0&1\\-1&0&1\\1&1&1\end{bmatrix}\right)\\ \equiv & \quad\boldsymbol{U}\boldsymbol{M}_{\mathbf{K}}\boldsymbol{V}=(\boldsymbol{V}^{\mathrm{T}}\otimes\boldsymbol{U})\mathrm{vec}(\boldsymbol{M}_{\mathbf{K}}),\end{aligned}$$

where

$$\boldsymbol{U} \equiv \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{M}_{\mathbf{K}} \equiv \begin{bmatrix} k_2 \\ k_1 \\ k_0 \end{bmatrix} \begin{bmatrix} k_5 & k_4 & k_3 \end{bmatrix}, \quad \boldsymbol{V} \equiv \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Further derive $\text{vec}(\boldsymbol{M}_{\mathbf{K}})$ as

$$\text{vec}(\boldsymbol{M}_{\mathbf{K}}) = \begin{bmatrix} k_2k_5 \\ k_1k_5 \\ k_0k_5 \\ k_2k_4 \\ k_1k_4 \\ k_0k_4 \\ k_2k_3 \\ k_1k_3 \\ k_0k_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_2k_5 \\ k_2k_4 \\ k_2k_3 \\ k_1k_5 \\ k_1k_4 \\ k_1k_3 \\ k_0k_5 \\ k_0k_4 \\ k_0k_3 \end{bmatrix} \equiv \boldsymbol{PK}$$

where $\boldsymbol{P}$ denotes the 9-by-9 permutation matrix in above equation. So

$$\begin{aligned} \boldsymbol{CK} = \quad & (\boldsymbol{V}^{\mathrm{T}} \otimes \boldsymbol{U})\text{vec}(\boldsymbol{M}_{\mathbf{K}}) = (\boldsymbol{V}^{\mathrm{T}} \otimes \boldsymbol{U})\boldsymbol{PK} \\ = \quad & \left(\begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}} \otimes \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{K} \\ = \quad & \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{K}. \end{aligned}$$

This is how to obtain the left side of (2.17). Solve (2.17) and obtain

$$\boldsymbol{K} = \quad \boldsymbol{C}^{-1}\boldsymbol{I}_{\bar{u},\bar{v}}$$

$$\begin{bmatrix} k_2k_5 \\ [5.7pt]k_2k_4 \\ [5.7pt]k_2k_3 \\ [5.7pt]k_1k_5 \\ [5.7pt]k_1k_4 \\ [5.7pt]k_1k_3 \\ [5.7pt]k_0k_5 \\ [5.7pt]k_0k_4 \\ [5.7pt]k_0k_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]-\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]-\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & -\frac{1}{2} & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]\frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]0 & \frac{1}{2} & 0 & 0 & -1 & 0 & 0 & \frac{1}{2} & 0 \\ [6pt]0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ [3pt]0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} I(\bar{u}-1,\bar{v}-1) \\ [5.7pt]I(\bar{u},\bar{v}-1 \\ [5.7pt]I(\bar{u}+1,\bar{v}- \\ [5.7pt]I(\bar{u}-1,\bar{v} \\ [5.7pt]I(\bar{u},\bar{v}) \\ [5.7pt]I(\bar{u}+1,\bar{v} \\ [5.7pt]I(\bar{u}-1,\bar{v}+ \\ [5.7pt]I(\bar{u},\bar{v}+1 \\ [5.7pt]I(\bar{u}+1,\bar{v}+ \end{bmatrix}$$

So based on the quadratic approximation assumption, the function value $I(u, v)$ can be computed by interpolating function values $I(\bar{u}+i,\bar{v}+j)$ $(i \in \{-1,0,+1\}$, $j \in \{-1,0,+1\})$ as

$$\begin{aligned} I(u,v) = \quad & \Delta u^2\Delta v^2 k_2k_5 + \Delta u^2\Delta v k_2k_4 + \Delta u^2 k_2k_3 \\ & + \Delta u\Delta v^2 k_1k_5 + \Delta u\Delta v k_1k_4 + \Delta u k_1k_3 + \Delta v^2 k_0k_5 + \Delta v k_0k_4 + k_0k_3 \\ = \quad & ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1])\boldsymbol{K}. \end{aligned} \tag{2.19}$$

Substitute (2.18) into (2.19) and obtain

$$\begin{aligned} I(u,v) = \quad & ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1])\boldsymbol{C}^{-1}\boldsymbol{I}_{\bar{u},\bar{v}} \\ = \quad & ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1]) \\ & \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]-\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]-\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & -\frac{1}{2} & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]\frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]0 & \frac{1}{2} & 0 & 0 & -1 & 0 & 0 & \frac{1}{2} & 0 \\ [6pt]0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ [6pt]0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} I(\bar{u}-1,\bar{v}-1) \\ [5.7pt]I(\bar{u},\bar{v}-1) \\ [5.7pt]I(\bar{u}+1,\bar{v}-1) \\ [5.7pt]I(\bar{u}-1,\bar{v}) \\ [5.7pt]I(\bar{u},\bar{v}) \\ [5.7pt]I(\bar{u}+1,\bar{v}) \\ [5.7pt]I(\bar{u}-1,\bar{v}+1) \\ [5.7pt]I(\bar{u},\bar{v}+1) \\ [5.7pt]I(\bar{u}+1,\bar{v}+1) \end{bmatrix} \end{aligned}$$

namely

$$I(u,v) = \quad ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1])$$
$$\begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]-\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ [6pt]0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]-\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & -\frac{1}{2} & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]\frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{4} & 0 & \frac{1}{4} \\ [6pt]0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ [6pt]0 & \frac{1}{2} & 0 & 0 & -1 & 0 & 0 & \frac{1}{2} & 0 \\ [6pt]0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ [6pt]0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \bar{I}\left(\bar{u}-1,\bar{v}-1\right) \\ [5.7pt]\bar{I}\left(\bar{u},\bar{v}-1\right) \\ [5.7pt]\bar{I}\left(\bar{u}+1,\bar{v}-1\right) \\ [5.7pt]\bar{I}\left(\bar{u}-1,\bar{v}\right) \\ [5.7pt]\bar{I}\left(\bar{u},\bar{v}\right) \\ [5.7pt]\bar{I}\left(\bar{u}+1,\bar{v}\right) \\ [5.7pt]\bar{I}\left(\bar{u}-1,\bar{v}+1\right) \\ [5.7pt]\bar{I}\left(\bar{u},\bar{v}+1\right) \\ [5.7pt]\bar{I}\left(\bar{u}+1,\bar{v}+1\right) \end{bmatrix} . \quad (2$$

Matlab code for computation of the matrices $\mathbf{C}$ and $\mathbf{C}^{-1}$ in above analysis is given as follows.[5]

SolveBiquadraticInterpolation.m

```
%% Solve Biquadratic Interpolation Coefficients
du = [-1; 0; 1]; dv = du';
U = [du.^2, du, du.^0]; V = [dv.^2; dv; dv.^0];
% vec(U * [k2; k1; k0]*[k5, k4, k3] * V) = (V' K.product U) * P
    * K
% K = [k2 k5; k2 k4; k2 k3; k1 k5; k1 k4; k1 k3; k0 k5; k0 k4;
    k0 k3]
%   = vec([k5; k4; k3]*[k2, k1, k0])
idx = reshape(transpose(reshape([1:9],3,[])),1,[]); P = zeros
    (3,3);
for r = 1:9   P(r,idx(r)) = 1; end % Permutation matrix

fprintf('P = '); P
fprintf('V^T kronecker_product U = '); KroneckerProduct(V', U)
fprintf('C = '); C = KroneckerProduct(V', U)*P
fprintf('inv(C) = '); inv(C)
% Iuv = [I(u-1,v-1); I(u,v-1); I(u+1,v-1); ...
%       I(u-1,v); I(u,v); I(u+1,v); I(u-1,v+1); I(u,v+1); I(u+1,
    v+1)]
% K = inv(C) * Iuv

%% Kronecker product
function Y = KroneckerProduct(X1, X2)
  H1 = size(X1,1); W1 = size(X1,2); H2 = size(X2,1); W2 = size(
    X2,2);
  Y = zeros(H1*H2, W1*W2);
  for r = 1:H1
     for c = 1:W1
         Y((r-1)*H2+1:r*H2, (c-1)*W2+1:c*W2) = X1(r,c)*X2;
     end
  end
end
```

2.1.4 Image Resolution Enhancement

The method of image interpolation enables one to indirectly estimate and analyse visual properties at arbitrary locations in the image area, not limited to integer (discrete) locations of the image. A direct application of image interpolation is *image resolution enhancement* , namely to transform an original low-resolution image into a new image of the same image content but with higher resolution.

Given an original image with low resolution. Represent it as $I(u, v)$ $(u \in \{1, \cdots, W\}$, $v \in \{1, \cdots, H\})$, where W denotes the image width and H denotes the image height. One intends to enhance the image resolution by a scaling factor s (say $s = 13$) and generate a new image. Represent the new image as $I_N(u_N, v_N)$ $(u_N \in \{1, \cdots, W_N \equiv sW\}$, $v_N \in \{1, \cdots, H_N \equiv sH\})$, where W_N and H_N denote respectively the width and height of the new image with higher resolution.

To generate the new image I_N from the original image I, let

$$I_N(cs, rs) = I(c, r), \quad (c \in \{1, \cdots, W\}, \quad r \in \{1, \cdots, H\})$$

or $\forall \quad 1 \leqslant u_N \leqslant W_N, 1 \leqslant v_N \leqslant H_N$,

$$I_N(u_N, v_N) = I(u_N/s, v_N/s), \quad (u_N \mod s = 0, \quad v_N \mod s = 0). \tag{2.21}$$

For any pixel of I_N not determined by (2.21), let the pixel value be computed by interpolating pixel values of its neighbours. For example, suppose the method of quadratic interpolation presented in Sect. 2.1.3 is adopted.

Consider a generic pixel $[u_N;v_N]^{\mathrm{T}}$ of I_N. Following (2.19) and (2.20), compute

$$\begin{aligned} u =& \quad u_N/s, \\ v =& \quad v_N/s, \\ \Delta u =& \quad u - \bar{u}, \\ \Delta v =& \quad v - \bar{v}, \end{aligned}$$

and

$$\begin{aligned} & I_N(u_N, v_N) \\ = & ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1]) \boldsymbol{C}^{-1} \boldsymbol{I}_{\bar{u},\bar{v}} \\ = & ([\Delta u^2 \quad \Delta u \quad 1] \otimes [\Delta v^2 \quad \Delta v \quad 1]) \\ & \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ {[6pt]} -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ {[6pt]} 0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} & 0 & 0 & 0 \\ {[6pt]} -\frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & -\frac{1}{2} & -\frac{1}{4} & 0 & \frac{1}{4} \\ {[6pt]} \frac{1}{4} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{4} & 0 & \frac{1}{4} \\ {[6pt]} 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ {[6pt]} 0 & \frac{1}{2} & 0 & 0 & -1 & 0 & 0 & \frac{1}{2} & 0 \\ {[6pt]} 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ {[6pt]} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} I(\bar{u}-1, \bar{v}-1) \\ {[5.7pt]} I(\bar{u}, \bar{v}-1) \\ {[5.7pt]} I(\bar{u}+1, \bar{v}-1) \\ {[5.7pt]} I(\bar{u}-1, \bar{v}) \\ {[5.7pt]} I(\bar{u}, \bar{v}) \\ {[5.7pt]} I(\bar{u}+1, \bar{v}) \\ {[5.7pt]} I(\bar{u}-1, \bar{v}+1) \\ {[5.7pt]} I(\bar{u}, \bar{v}+1) \\ {[5.7pt]} I(\bar{u}+1, \bar{v}+1) \end{bmatrix}. \end{aligned} \tag{2.22}$$

Matlab code for image resolution enhancement via biquadratic interpolation is given as follows.

BiquadraticInterpolation.m

```
%% Biquadratic Interpolation
function imgI = BiquadraticInterpolation(imgOrg, sc) % gray-
    level image
   % Compute biquadratic interpolation coefficients
   img = double(imgOrg); du = [-1; 0; 1]; dv = du’;
   U = [du.^2, du, du.^0]; V = [dv.^2; dv; dv.^0];
   idx = reshape(transpose(reshape([1:9],3,[])),1,[]); P = zeros
    (3,3);
   for r = 1:9 P(r,idx(r)) = 1; end % Permutation matrix
   C = kron(V’, U)*P; invC = inv(C);

   % Generate interpolated image with high resolution
   H = size(img,1); W = size(img,2); H2 = H*sc; W2 = W*sc;
   imgI = uint8(zeros(H2, W2));
   for r = 1:H2
       for c = 1:W2
           u = c/sc; ub = round(u); ub = max(2, min(W-1, ub));
    du = u - ub;
           v = r/sc; vb = round(v); vb = max(2, min(H-1, vb));
    dv = v - vb;
           Iuv = reshape(img(vb-1:vb+1, ub-1:ub+1)’,[],1);
           duv = kron([du^2, du, 1], [dv^2, dv, 1]);
           Itmp = duv*invC*Iuv; imgI(r,c) = max(0, min(255,
    round(Itmp)));
       end
   end
end
```

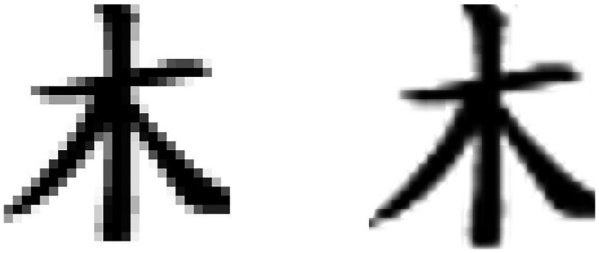

(a) (b)

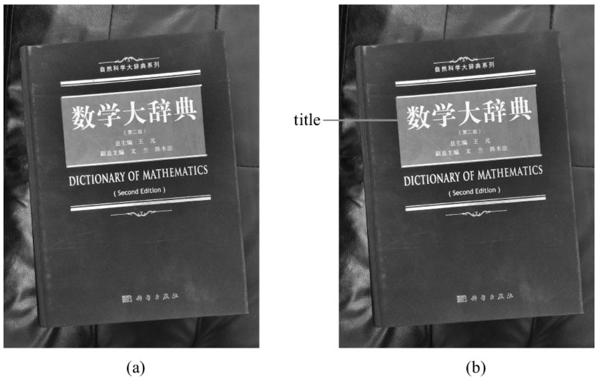
Fig. 2.6 Biquadratic interpolation **a** original image with resolution 27-by-25; **b** interpolated image with higher resolution 351-by-325

An example of image resolution enhancement via biquadratic interpolation is demonstrated in Fig. 2.6. Figure 2.6a shows the original low-resolution image, whereas Fig. 2.6b shows the new image obtained from the original image via biquadratic interpolation. Compared with the Chinese character “wood” on the left, the Chinese character “wood” on the right obviously has refined visual effect.

2.2 Feature Points

An image usually possesses a huge amount of information. Take a gray-level image of moderate size 640-by-480 as example, which contains $640 \times 480 = 307\,200$ pixels. Suppose each pixel is represented by one byte, then the image contains 307 200 bytes that can encode about fifty thousand Chinese names—The name list of the entire population of a town—Possessing plenty of information is right an advantage of images.[6] Sometimes, an image is more informative than a thousand of words.

On the other hand, possessing plenty of information is right a drawback of images, because the absolutely most part of information contained in an image is usually useless in concrete applications. In other words, useful information usually holds an absolutely small part of information in an image. For example, consider the photo of a book illustrated in Fig. 2.7. Suppose what one wants to know is the book title in Chinese.



(a)

(b)

Fig. 2.7 Book **a** original photo of the book; **b** indicated book title

In this concrete case, from the perspective of computer vision, what one really needs to care, namely the objects of interest, are those bright shapes corresponding to Chinese characters of the book title (indicated in Fig. 2.7b). The objects of interest only hold a small part of the entire image. Consequently, the huge amount of useless information contained in an image tends to “dilute” processing ability actually dedicated to objects of interest.

In practical applications, instead of treating and processing all image pixels equivalently, a computer vision system can perform preliminary image processing to extract certain **feature set** that is highly related to objects of interest. Then for subsequent processing, the computer vision system can focus on the feature set without wasting computational resources on a huge amount of useless information contained in images.

Points are basic elements of geometry. Various geometric entities are essentially different sets of points in certain space. Similarly, a feature set consists of **feature points**. In this chapter, the author presents a number of representative methods of **feature point extraction** or **feature extraction** for short.

2.2.1 Thresholding: Pixel-Wise Point Extraction

A representative method of feature point extraction is based on the natural idea of **binary categorization**: The image pixels are categorized into two mutually-exclusive sets of pixels (or points); one called **foreground** is related to or regarded as objects of interest, whereas the other called **background** is not related to or not regarded as objects of interest.

Such representative method that performs binary categorization in pixel-wise way is called **thresholding**. Generically speaking, traverse image pixels one by one. For each traversed pixel, check whether the pixel satisfies certain foreground condition: If satisfied, assign the pixel to the foreground set; otherwise, assign the pixel to the background set. The generic thresholding algorithm in pseudo code is given below.

Algorithm: Thresholding

```
FOR EACH pixel IN image
    IF pixel SATISFIES the foreground condition
        Assign pixel to the foreground set
    ELSE
        Assign pixel to the background set
    END
END
```

The foreground condition can be of various kinds. For examples (gray-level images)

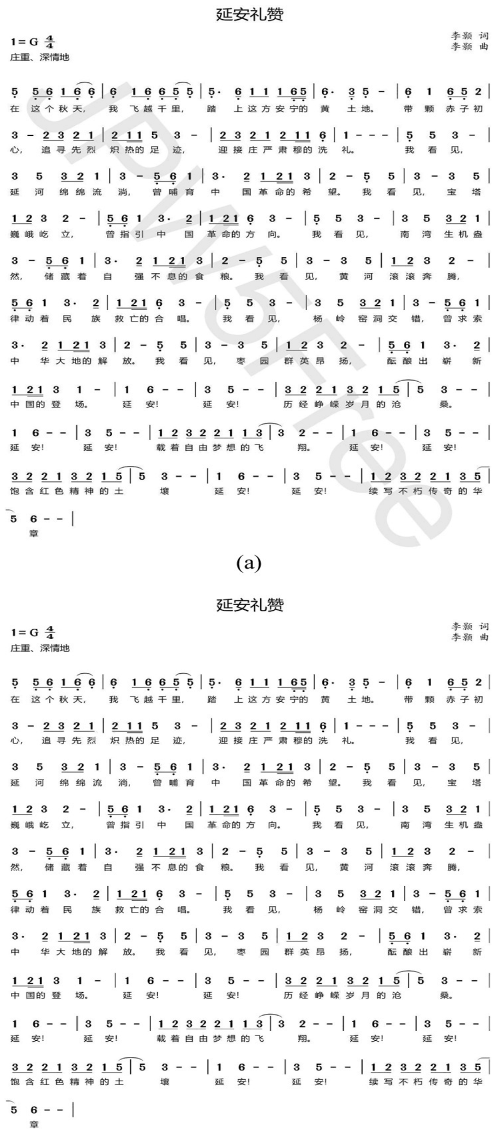
- Foreground condition example 1: The pixel value (intensity) is above certain threshold.
- Foreground condition example 2: The pixel value is below certain threshold.
- Foreground condition example 3: The pixel value is between certain threshold and another threshold.
- Foreground condition example 4: The pixel value is in an interval or in another interval.

For other examples (color images)

- Foreground condition example 1: The pixel red (or green, blue) channel value is above certain threshold.
- Foreground condition example 2: The pixel red (or green, blue) channel value is below certain threshold.
- Foreground condition example 3: The pixel red channel value is above certain threshold, whereas the pixel green and blue channel values are below other thresholds.
- Foreground condition example 4: Certain combination of the pixel red, green, blue channel values is between certain threshold and another threshold.

2.2.2 Application of Image Thresholding: Watermark Removal

An application of image thresholding in daily life is *watermark removal* for text-type documents. Watermarks are usually phantom light shadows embedded into the background of an image. Some document editing software may add watermarks to document images that it generates. For example, a software “JP-Word” for numbered musical notation editing is available for partially free usage, yet the free version of the software “JP-Word” adds the watermark “JPW5Free” to music score images (including PDF) that it generates, as illustrated in Fig. 2.8a, which displays the music score of “Tribute to Yan-An”.[7]



(a)

(b)

Fig. 2.8 Music score images **a** score image with watermark; **b** score image with watermark removed

To remove the document watermark illustrated in Fig. 2.8a, one may resort to image thresholding. The music score in black text is the intended foreground set that consists of pixels with low pixel values (gray-level intensities), whereas the undesirable watermark and the white empty area form the background that consists of pixels with higher pixel values. So one can set the foreground condition simply as

- Foreground condition: The pixel value is below a threshold (e.g. 220).

The score image with watermark removed is demonstrated in Fig. 2.8b. Matlab code for watermark removal via image thresholding is given as follows.

RemoveWatermark.m

```
%% Remove watermark
img = rgb2gray(imread(imgName));
H = size(img,1); W = size(img,2); imgOrg = img;
pixelT = 220; % Pixel value threshold
% Traverse image pixels
for r = 1:H
    for c = 1:W
        if (img(r,c) <= pixelT)
            fprintf('Do nothing to the pixel ...\n');
        else
            % Set non-foreground to white background
            img(r,c) = 255;
        end
    end
end
% Above loop code can be implemented in Matlab simply as
% idx = find(img > pixelT); img(idx) = 255;
figure(1), subplot(1,2,1), imshow(imgOrg); subplot(1,2,2),
    imshow(img);
```

2.2.3 Adaptive Thresholding

Selection of thresholds depends rather on *ad hoc* applications and there is no universal rule for selection of thresholds in all applications. Sometimes, an appropriate threshold may be set empirically according to a priori knowledge, such as in the application of watermark removal via image thresholding. In general scenarios, it is more desirable that an appropriate threshold would be estimated adaptively, depending dynamically on image statistics in concrete applications [2].

An effective idea is to compare each pixel with an average of its surrounding pixels. Suppose the image primarily contains background pixels (to be categorized as "white") whereas the foreground pixels (to be categorized as "black") are rather splashed in the image. The adaptive thresholding algorithm in pseudo code is given below.

Algorithm: Adaptive Thresholding

```
FOR EACH pixel IN image
    Compute the value average of pixel's surrounding pixels
    IF pixel value is $t$ percent lower than the value average
        Assign pixel to the foreground (black) set
    ELSE
        Assign pixel to the background (white) set
    END
END
```

For a generic pixel namely a generic point $\boldsymbol{P} = [u, v]^{\mathrm{T}}$ with integer u and v in the image area, the value average of the surrounding pixels can be computed by traversing them one by one, as

$$E[I(u-s:u+s, v-s:v+s)] = \frac{1}{(2\,s+1)^2} \sum\nolimits_{r=v-s}^{v+s} \sum\nolimits_{c=u-s}^{u+s} I(c,r),$$

where s denotes half the size of the surrounding area of each pixel. However, such naive implementation is not efficient at all. To make implementation of the adaptive thresholding algorithm more efficient, one can take advantage of the **integral image**, refer to Appendix A for details.

Following (A.4), one has a much more efficient way to compute the value average of a pixel's surrounding pixels:

$$\begin{aligned} E[\quad I(u-s:u+s, v-s:v+s)] = \tfrac{1}{(2s+1)^2}&[I_S(u+s, v+s) \\ -I_S(u+s, v-s-1) - I_S(u-s-1, v+s) &+ I_S(u-s-1, v-s-1)]. \end{aligned} \tag{2.23}$$

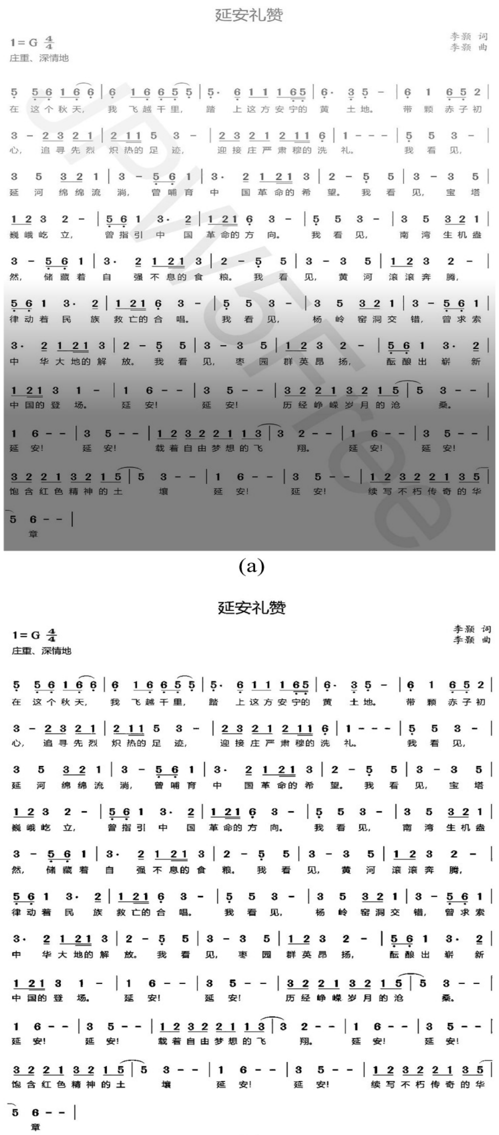
Realization of the integral image based technique incurs an extra computational cost of pre-processing for integral image computation. Since integral image computation requires just one traversal of the image, the extra computational cost of pre-processing for integral image computation is negligible when compared with the computational cost that the integral image based technique saves.

Matlab code for adaptive thresholding that takes advantage of the integral image is given as follows—Refer to Appendix A for the scripts **IntegralImageComputation.m** and **GetIntegralImageRect.m**.

AdaptiveThresholding.m

```
%% Adaptive Thresholding
function imgB = AdaptiveThresholding(img, s, t)
    [H,W] = size(img); imgB = zeros(H,W);
    if (nargin<2) s=round(min(H,W)/16); t=0.15; end
    % First traversal of image pixels
    imgS = IntegralImageComputation(img);
    % Second traversal of image pixels
    for r = 1:H
        for c = 1:W
            u1=max(0,c-s-1); v1=max(0,r-s-1);
            u2=min(W,c+s); v2=min(H,r+s); a=(u2-u1)*(v2-v1);
            av = GetIntegralImageRect(imgS,u1,v1,u2,v2)/a;
            if (img(r,c) < (1-t)*av)
                imgB(r,c) = 0; % Black foreground
            else
                imgB(r,c) = 255; % White background
            end
        end
    end
end
```

An application example of adaptive thresholding is demonstrated in Fig. 2.9. Figure 2.9a displays the same score “Tribute to Yan-An” and the same watermark “JPW5Free” as what Fig. 2.8a displays. However, compared with the score image illustrated in Fig. 2.8a, the score image illustrated in Fig. 2.9a is obtained as if under severely-uneven illumination.



(a)

(b)

Fig. 2.9 Adaptive thresholding **a** score image with watermark under severely-uneven illumination; **b** clear score image

In fact, in the score image illustrated in Fig. 2.9a, the “black” score foreground near the top of the image is even as bright as the “white” non-score background near the bottom of the image—Readers may have an illusion that the “black” score foreground near the top of the image seems to be still darker than the “white” non-score background near the bottom of the image. The illusion is actually caused by humans’ biological vision mechanism that strengthens brightness or intensity contrast. Refer to Appendix B for more details—Consequently, one cannot extract the clear score image simply by fixed thresholding.

On the other hand, the problem of extracting the clear score image from the undesirable original image illustrated in Fig. 2.9a can be effectively handled by adaptive thresholding. The extraction effect of adaptive thresholding is demonstrated inFigure 2.9b, which is realized by the Matlab code **AdaptiveThresholding.m** given above.

2.2.4 Non-Maximum Suppression: Extremum Point Extraction

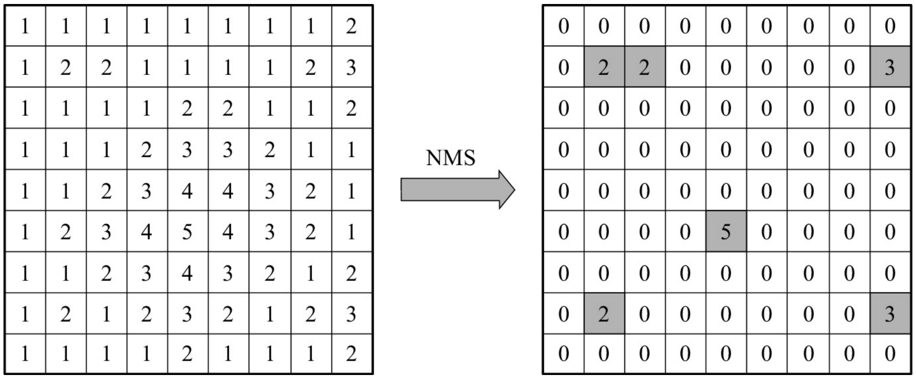
Extremum points, namely points that satisfy certain extremum condition, are a special kind of basic elements of geometry. Sometimes, one needs to constrain consideration of feature points to points that satisfy certain extremum condition. A commonly adopted extremum condition is possessing the locally maximum or minimum point value.

Possessing the locally maximum point value and possessing the locally minimum point value are essentially the same from the perspective of extrema analysis—If signs of point values are reversed, then points that possess locally minimum point values before sign reversal will become points that possess locally maximum point values after sign reversal —So the author constrains consideration of extrema to points that possess locally maximum point values.

A representative method of extremum point extraction is based on the idea of **non-maximum suppression**: Traverse the image, and for each pixel traversed, if its pixel value is smaller than at least one neighbour's pixel value, then suppress its pixel value to zero. After one traversal of the image in such way, only extremum points keep their pixel values whereas all other non-extremum points' pixel values are suppressed to zero. Matlab code for non-maximum suppression is given as follows.

NonMaximumSuppression.m

```
%% Non-Maximum Suppression
function imgNMS = NonMaximumSuppression(img)
    H = size(img,1)+2; W = size(img,2)+2; imgNMS = img;
    imgD = zeros(H,W); imgD(2:H-1,2:W-1) = double(img);
    iOff = [-H-1, -H, -H+1, -1, +1, H-1, H, H+1]; %
    Neighbourhood offsets
    for r = 1:H-2 % v
        for c = 1:W-2 % u
            i = c*H+r+1;
            for i2 = i + iOff
                if (imgD(i)<imgD(i2)) imgNMS(r,c) = 0; break;
    end
            end
        end
    end
end
```



1	1	1	1	1	1	1	1	2
1	2	2	1	1	1	1	2	3
1	1	1	1	2	2	1	1	2
1	1	1	2	3	3	2	1	1
1	1	2	3	4	4	3	2	1
1	2	3	4	5	4	3	2	1
1	1	2	3	4	3	2	1	2
1	2	1	2	3	2	1	2	3
1	1	1	1	2	1	1	1	2

0	0	0	0	0	0	0	0	0
0	2	2	0	0	0	0	0	3
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	5	0	0	0	0
0	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	3
0	0	0	0	0	0	0	0	0

Fig. 2.10 Non-maximum suppression

Once non-maximum suppression is performed, extremum points can be conveniently collected by another traversal of the image. An example of non-maximum suppression is demonstrated in Fig. 2.10. The left matrix of data represents an image with its original pixel values, whereas the right matrix of data represents the obtained image after non-maximum suppression.

Readers may wonder: during traversal of the image, why not directly extract extremum points with locally maximum pixel values? The algorithm in pseudo code that realizes such idea is given below.

Algorithm: Direct maximum extraction

```
FOR EACH pixel IN image
    IF pixel value is larger than all neighbouring pixel values
        Extract pixel as extremum point
    END
END
```

When only an individual static extremum condition is handled in practical applications, the direct maximum extraction algorithm may indeed be adopted, because it reflects almost the same logic as that of the non-maximum suppression algorithm.

On the other hand, when multiple extrema conditions that may vary dynamically are handled in practical applications, the non-maximum suppression algorithm tends to have merits. By non-maximum suppression, the computer vision system does not rely on determining whether a pixel is an extremum point at one strike, which may be difficult sometimes, especially in presence of dynamic extrema conditions. Instead, the computer vision system focuses on suppressing points that are definitely not extremum points during non-maximum suppression, which can always be realized conveniently, be extrema conditions static or dynamic. After all necessary processing, points that can survive from non-maximum suppression by various extrema conditions are naturally extremum points and can be collected by a final traversal of the image—It is like for sport games such as football, the committee may not use one set of rules to determine winner teams. Instead, the committee tends to use some set of rules to sift out teams that are definitely not winners, use another set of rules to sift out among survived teams ones that are definitely not winners, then use a third set of rules to sift out among further survived teams ones that are definitely not winners either, and so on. The teams that survive finally are naturally winners. After each round of sifting, the survived teams cannot be predicted a priori; in other words, they can be regarded as dynamic factors for “extrema” extraction. Such process of determining winners for sport games well reflects spirit of non-maximum suppression.

Non-maximum Suppression Among a Sparse Set of Points

Sometime one may not need to perform non-maximum suppression over the entire image, but instead to perform non-maximum suppression among a set of points that are much lesser than the set of all pixels in the image. In other words, one may need to perform non-maximum suppression among only a sparse set of points in the image. Matlab code for such special purpose of non-maximum suppression is given as follows.

NonMaximumSuppressionPoints.m

```
%% Non-Maximum Suppression among a set of image points
function ptNMS = NonMaximumSuppressionPoints(pt)
    H = max(pt(2,:))+1; W = max(pt(1,:))+1;
    img = zeros(H,W); ptNMS = pt;
    for k = 1:size(pt,2)
        img(pt(2,k),pt(1,k)) = pt(3,k);
    end
    iOff = [-H-1, -H, -H+1, -1, +1, H-1, H, H+1]; %
    Neighbourhood offsets
    for k = 1:size(pt,2)
        i = H*(pt(1,k)-1)+pt(2,k);
        for i2 = i+iOff
            if (img(i)<img(i2)) ptNMS(3,k) = 0; break; end
        end
    end
    idx = find(ptNMS(3,:)>0); ptNMS = ptNMS(:,idx);
end
```

Non-maximum Suppression of the Three-Dimensional Tensor Image

From time to time in practical applications, one needs to perform non-maximum suppression over a three-dimensional tensor image. Simply speaking, a three-dimensional tensor image may be treated as a stack of conventional two-dimensional images; it can be denoted by a trivariate function $\bar{I}(u, v, k)$, where k denotes the order index of the image in the stack.

For a generic point (u, v, k) in the three-dimensional tensor image, its neighbourhood in the tensor image is defined as

$$\bar{I}(u-1:u+1, v-1:v+1, k-1:k+1)$$

namely

$$\{\bar{I}(u+\delta u, v+\delta v, k+\delta k) | \delta u, \delta v, \delta k \in \{-1, 0, +1\}\}.$$

In other words, for non-maximum suppression of the three-dimensional tensor image, each point (u, v, k) needs to be compared with the other twenty-six points in its neighbourhood. If it has a value smaller than any of the other twenty-six points,[8] then it is suppressed to zero. Matlab code for non-maximum suppression of the three-dimensional tensor image is given as follows.

NonMaximumSuppression3D.m

```
%% Non-Maximum Suppression of three-dimensional tensor image
function imgNMS = NonMaximumSuppression3D(img)
    [H,W,T] = size(img); imgNMS = zeros(H,W,T);
    % Omit boundary by default
    imgNMS(2:H-1,2:W-1,2:T-1) = img(2:H-1,2:W-1,2:T-1);
    iOff=[-H-1,-H,-H+1,-1,0,+1,H-1,H,H+1]; iOff=[iOff,iOff-H*W,
    iOff+H*W];
    rm = [2:H-1]'; cm = [2:W-1]; ts = zeros(H,W,length(iOff));
    im = reshape(rm*ones(size(cm))+H*ones(size(rm))*(cm-1),1,[])
    ;
    for k = 2:T-1
        i = im + (k-1)*H*W; ts(:,:,:) = 0;
        for i2 = 1:length(iOff) % Neighbourhood offset tensor
            ts(im+(i2-1)*H*W) = img(i+iOff(i2));
        end
        Imax = max(ts,[],3); i = find(Imax<0); Imax(i) = 0;
        Imin = min(ts,[],3); i = find(Imin>0); Imin(i) = 0;
        i = find(imgNMS(:,:,k)<Imax & imgNMS(:,:,k)>Imin);
        imgNMS(i+(k-1)*H*W) = 0;
    end
end
```

2.2.5 Dilation and Erosion: Contour Extraction

Dilation and **erosion** are two other representative methods for geometric operation of feature sets. Given an image whose pixels are categorized into two exclusive sets of pixels namely a foreground set and a background set. Intuitively speaking, from the perspective of the foreground set, dilation means assimilating the background set's boundary part that touches the foreground set as foreground, whereas erosion means peeling the foreground set's boundary part that touches the background set. Suppose the 8-neighbourhood illustrated in Fig. 2.11b is treated as neighbourhood for each pixel, the mutual relationship between dilation and erosion is illustrated in Fig. 2.12.

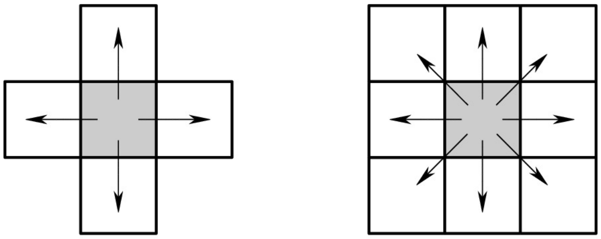

(a) (b)

Fig. 2.11 Neighbourhood **a** 4-neighbourhood; **b** 8-neighbourhood

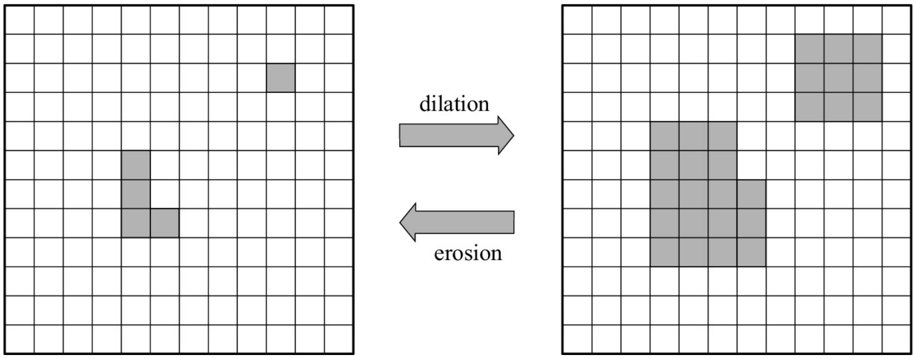


Fig. 2.12 Dilation and erosion

Besides above intuitive demonstration, the idea of dilation in more operational terms is: Traverse the image, and for each pixel traversed, if it belongs to the background set but it has a neighbour belonging to the foreground set, then assign it to the foreground set. Otherwise, still assign it to the background set. In other words, for each pixel, if it has a foreground pixel in its neighbourhood (including itself), then assign it to the foreground set; otherwise, assign it to the background set.

Similarly, the idea of erosion in more operational terms is: Traverse the image, and for each pixel traversed, if it belongs to the foreground set but it has a neighbour belonging to the background set, then assign it to the background set. Otherwise, still assign it to the foreground set. In other words, for each pixel, if it has a background pixel in its neighbourhood (including itself), then assign it to the background set; otherwise, assign it to the foreground set. The dilation and erosion algorithms in pseudo code are given below.

Algorithm: Dilation

```
FOR EACH pixel IN image
    IF pixel has a foreground pixel in the neighbourhood
        Assign pixel to the foreground set
    ELSE
        Assign pixel to the background set
    END
END
```

Algorithm: Erosion

```
FOR EACH pixel IN image
    IF pixel has a background pixel in the neighbourhood
        Assign pixel to the background set
    ELSE
        Assign pixel to the foreground set
    END
END
```

Figure 2.12 may give readers an illusion that dilation and erosion form a pair of mutually-inverse transform, namely after a pair of dilation and erosion, the obtained image will always return to the original image. If really so, it seems useless to perform dilation and erosion together in pair. Fortunately, it is not so, performing dilation and erosion together will form a useful effect of contour extraction, as demonstrated in Fig. 2.13.

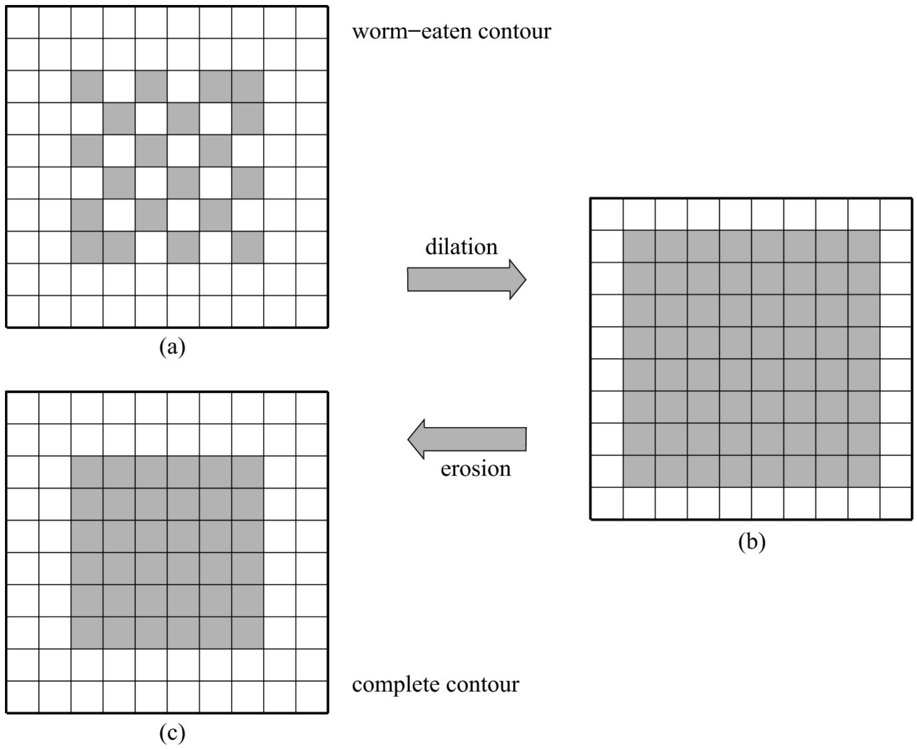


Fig. 2.13 Dilation and erosion for contour extraction

Figure 2.13a illustrates an image that contains a set of points with worm-eaten contour. First, apply dilation to the image and obtain an intermediate image illustrated in Fig. 2.13b. Then apply erosion to the intermediate image to recover the size of the original set of points, as illustrated in Fig. 2.13c. As shown in Fig. 2.13, after such pair of dilation and erosion, a new set of points that has the same size as the original set of points and has a complete contour is obtained.

Matlab code for dilation and erosion of a binary image in which 0 means foreground is given as follows.

Dilate.m

```
%% Dilate an image : 0 means foreground
function imgR = Dilate(img)
    H = size(img,1); W = size(img,2); imgR = img;
    kOffset = [-H-1, -H, -H+1, -1, +1, +H-1, +H, +H+1];
    for r = 2:H-1 % Neglect image boundary part
        for c = 2:W-1
            kb = (c-1)*H+r;
            for k = kb + kOffset
                if (0 == img(k)) imgR(kb) = 0; break; end
            end
        end
    end
end
```

Erode.m

```
%% Erode an image : 0 means foreground;
function imgR = Erode(img)
    H = size(img,1); W = size(img,2); imgR = img;
    kOffset = [-H-1, -H, -H+1, -1, +1, +H-1, +H, +H+1];
    for r = 2:H-1 % Neglect image boundary part
        for c = 2:W-1
            kb = (c-1)*H+r;
            for k = kb + kOffset
                if (img(k)>0) imgR(kb) = 255; break; end
            end
        end
    end
end
```

2.3 Edge Points

Humans’ biological vision systems possess a mechanism that strengthens brightness or intensity contrast (refer to Appendix B), so humans’ vision is rather sensitive to object edges which tend to cause considerable brightness contrast. Computer vision systems in many practical applications need to extract edges (namely sets of edge points) from images.

2.3.1 Gradient Image

Object edges tend to correspond to image locations with considerable pixel value contrast. Therefore, to extract edge points, one may extract image points with large gradients and treat such image points as candidate edge points for further processing.

Denote the image pixel value function as $I(u, v)$, and suppose the image height is H and the image width is W. The image pixel value function $I(u, v)$ is represented by the image pixel value matrix $\bar{I}(u,v)$ defined on the positive integer bivariate domain $\boldsymbol{I}_{\mathrm{D}}$, as in (2.2):

$$I(u,v) = \bar{I}(u,v),$$

where $\boldsymbol{I}_{\mathrm{D}}$ is specified in (2.1).

Consider the gradient function of $I(u, v)$, namely

$$\nabla I(u,v) = \begin{bmatrix} I_u(u,v) \\ I_v(u,v) \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial I(u,v)}{\partial u} \\ \frac{\partial I(u,v)}{\partial v} \end{bmatrix},$$

where $I_u(u,v) \equiv \frac{\partial I(u,v)}{\partial u}$ and $I_v(u,v) \equiv \frac{\partial I(u,v)}{\partial v}$ denote partial derivatives of $I(u, v)$ with respect to u and v respectively.

The discrete subset of $\nabla I(u,v)$ defined on $\boldsymbol{I}_{\mathrm{D}}$, denoted as $\nabla \bar{I}(u,v)$, forms a three-dimensional tensor:

$$\nabla \bar{I}(u,v) = \begin{bmatrix} I_u(u,v) \\ I_v(u,v) \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial I(u,v)}{\partial u} \\ \frac{\partial I(u,v)}{\partial v} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \end{bmatrix} \in \boldsymbol{I}_{\mathrm{D}}, \tag{2.24}$$

which is called the **gradient image** of the original image. The discrete subsets of the partial derivative functions $I_u(u,v)$ and $I_v(u,v)$ defined on $\boldsymbol{I}_{\mathrm{D}}$ form the u-part and v-part of the gradient image (namely the u-gradient image and the v-gradient image) respectively.

The gradient image $\nabla\bar{I}\,(u, v)$ is the focus for gradient analysis of the image $I(u, v)$. One can approximate computation of the gradient image via numeric methods. For example

$$\nabla\bar{I}\,(u, v) \approx \begin{bmatrix} \frac{I(u+\Delta u, v)-I(u,v)}{\Delta u} \\ [9pt]\,\frac{I(u,v+\Delta v)-I(u,v)}{\Delta v} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \end{bmatrix} \in \boldsymbol{I}_{\mathrm{D}}. \tag{2.25}$$

A better numeric approximation is

$$\nabla\bar{I}\,(u, v) \approx \begin{bmatrix} \frac{I(u+\Delta u, v)-I(u-\Delta u,v)}{2\Delta u} \\ [9pt]\,\frac{I(u,v+\Delta v)-I(u,v-\Delta v)}{2\Delta v} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \end{bmatrix} \in \boldsymbol{I}_{\mathrm{D}}. \tag{2.26}$$

Reasons are as follows: One may fix v and treat $I(u, v)$ as a one-dimensional function denoted as $I(u)$ for short (similar analysis can be made when u is fixed). Consider Taylor expansion of $I(u)$ about a generic u:

$$\begin{aligned} I(u+\Delta u) &= I(u) + I'(u)\Delta u + \tfrac{I''(u)}{2}\Delta u^2 + o(\Delta u^2), \\ I(u-\Delta u) &= I(u) - I'(u)\Delta u + \tfrac{I''(u)}{2}\Delta u^2 + o(\Delta u^2). \end{aligned}$$

So

$$\tfrac{I(u+\Delta u)-I(u)}{\Delta u} = I'(u) + \tfrac{I''(u)}{2}\Delta u + o(\Delta u) = I'(u) + o(1)$$

gives a first-order infinitesimal approximation of the derivative $I'(u)$, whereas

$$\tfrac{I(u+\Delta u)-I(u-\Delta u)}{2\Delta u} = I'(u) + o(\Delta u)$$

gives a second-order infinitesimal approximation of the derivative $I'(u)$. The latter is apparently better than the former.

A natural way to set Δu and Δv on the positive integer bivariate domain $\boldsymbol{I}_{\mathrm{D}}$ is to set $\Delta u = \Delta v = 1$. Then (2.26) becomes

$$\nabla\bar{I}\,(u, v) \approx \begin{bmatrix} \frac{I(u+1, v)-I(u-1,v)}{2} \\ [9pt]\,\frac{I(u,v+1)-I(u,v-1)}{2} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \end{bmatrix} \in \boldsymbol{I}_{\mathrm{D}}. \tag{2.27}$$

In other words, the gradient image $\nabla\bar{I}\,(u, v)$ is approximately obtained by performing two discrete convolutions on the image pixel value matrix $\bar{I}\,(u, v)$ using the different convolution templates $\begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix}$ and $\begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}^{\mathrm{T}}$ respectively:

$$\bar{I}_{\,u}(u, v) \approx \begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} * \bar{I}\,(u, v), \quad \bar{I}_{\,v}(u, v) \approx \begin{bmatrix} -\frac{1}{2} \\ [5pt]0 \\ \frac{1}{2} \end{bmatrix} * \bar{I}\,(u, v). \tag{2.28}$$

The convolution template $\begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix}$ represents the discrete bivariate function

$$\begin{bmatrix}\frac{1}{2} & 0 & -\frac{1}{2}\end{bmatrix} \equiv \begin{cases} -\frac{1}{2} & u=1, v=0 \\ {[9pt]}\,\frac{1}{2} & u=-1, v=0 \\ {[5pt]}0 & \text{otherwise} \end{cases}$$

whereas the convolution template $\begin{bmatrix}-\frac{1}{2} & 0 & \frac{1}{2}\end{bmatrix}^{\mathrm{T}}$ represents the discrete bivariate function

$$\begin{bmatrix}-\frac{1}{2} \\ 0 \\ \frac{1}{2}\end{bmatrix} \equiv \begin{cases} -\frac{1}{2} & u=0, v=1 \\ {[9pt]}\,\frac{1}{2} & u=0, v=-1 \\ {[5pt]}0 & \text{otherwise} \end{cases}$$

The gradient image $\nabla \bar{I}(u,v)$ may also be approximately computed as

$$\nabla \bar{I}(u,v) \approx \begin{bmatrix} \dfrac{\begin{array}{c} I(u+1,v+1)+2I(u+1,v)+I(u+1,v-1)- \\ I(u-1,v+1)-2I(u-1,v)-I(u-1,v-1) \end{array}}{8} \\ {[9pt]}\,\dfrac{\begin{array}{c} I(u+1,v+1)+2I(u,v+1)+I(u-1,v+1)- \\ I(u+1,v-1)-2I(u,v-1)-I(u-1,v-1) \end{array}}{8} \end{bmatrix}, \quad \begin{bmatrix} u \\ v \end{bmatrix} \in \boldsymbol{I}_{\mathrm{D}}, \tag{2.29}$$

or from the convolution perspective as

$$\bar{I}_u(u,v) \approx \begin{bmatrix} \frac{1}{8} & 0 & -\frac{1}{8} \\ {[9pt]}\,\frac{1}{4} & 0 & -\frac{1}{4} \\ {[9pt]}\,\frac{1}{8} & 0 & -\frac{1}{8} \end{bmatrix} * \bar{I}(u,v), \quad \bar{I}_v(u,v) \approx \begin{bmatrix} -\frac{1}{8} & -\frac{1}{4} & -\frac{1}{8} \\ {[5pt]}0 & 0 & 0 \\ {[5pt]}\,\frac{1}{8} & \frac{1}{4} & \frac{1}{8} \end{bmatrix} * \bar{I}(u,v). \tag{2.30}$$

The method described in (2.29) and (2.30) is called the **Sobel** method.

Matlab code for gradient image computation via (2.27) and (2.29) is given as follows.

GradientImage.m

```
%% Compute the gradient image
function [Iu, Iv] = GradientImage(img)
    H = size(img,1); W = size(img,2);
    imgD = zeros(H+2, W+2); imgD(2:H+1,2:W+1) = double(img);
    Iu = (imgD(2:H+1,3:W+2)-imgD(2:H+1,1:W))/2; % u-gradient
    image
    Iv = (imgD(3:H+2,2:W+1)-imgD(1:H,2:W+1))/2; % v-gradient
    image
end
```

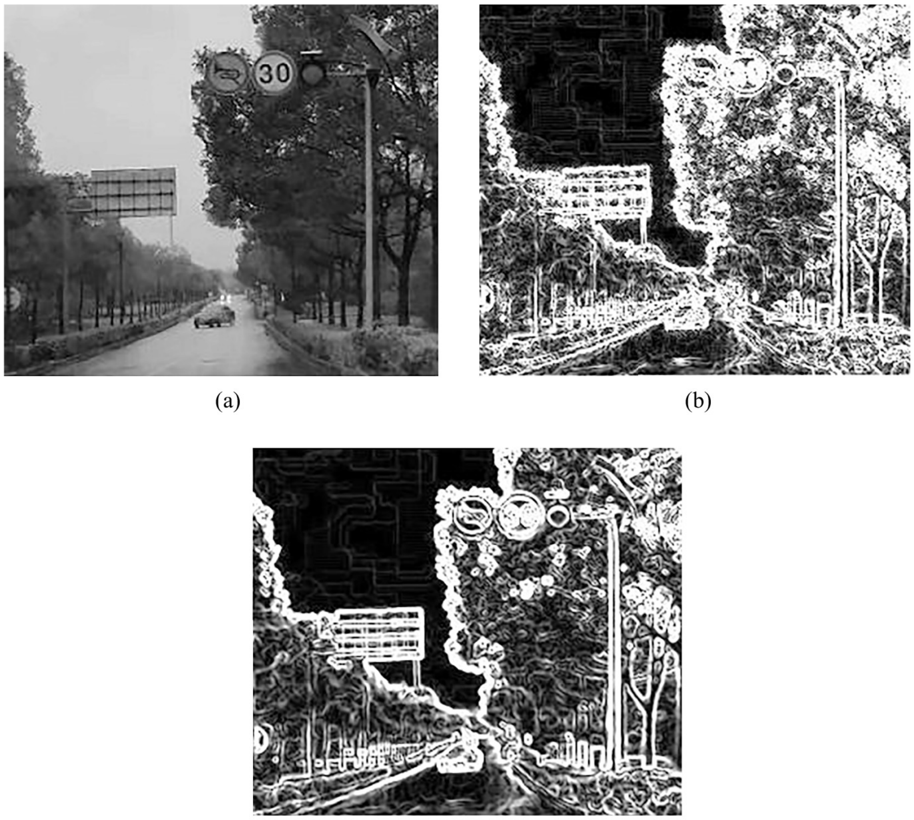
GradientImageSobel.m

```
%% Compute the gradient image using the Sobel method
function [Iu, Iv] = GradientImageSobel(img)
    H = size(img,1); W = size(img,2);
    imgD = zeros(H+2, W+2); imgD(2:H+1,2:W+1) = double(img);
    Iu = (imgD(1:H,3:W+2)+2*imgD(2:H+1,3:W+2)+imgD(3:H+2,3:W+2)
    ...
        -imgD(1:H,1:W)-2*imgD(2:H+1,1:W)-imgD(3:H+2,1:W))/8; % u
    -gradient
    Iv = (imgD(3:H+2,1:W)+2*imgD(3:H+2,2:W+1)+imgD(3:H+2,3:W+2)
    ...
        -imgD(1:H,1:W)-2*imgD(1:H,2:W+1)-imgD(1:H,3:W+2))/8; % v
    -gradient
end
```

Gradient computation is rather susceptible to image noises. A preliminary step of image filtering (see Appendix C) usually precedes gradient computation. For example, the commonly-adopted Gaussian filtering (see Sect. C.4.2 of Appendix C) may be applied first to smooth the original image and then the gradient image is computed from the smoothed image.

An example of gradient image effects is illustrated in Fig. 2.14. Figure 2.14a is the original image that shows a traffic scenario. Figure 2.14b illustrates the gradient image that is computed direct from the original image. Figure 2.14c illustrates the gradient image that is computed from the original image smoothed via Gaussian filtering.[9]

As shown in Fig. 2.14, the gradient image illustrated in the top-right sub-figure is suffused with noisy locations of large gradients. In contrast, the gradient image illustrated in the bottom sub-figure is much cleaner.



(a)

(b)

(c)

Fig. 2.14 Gradient image **a** original image; **b** gradient image without image filtering; **c** gradient image after image filtering

2.3.2 Directional Non-maximum Suppression

As demonstrated in Fig. 2.14, the gradient image can intuitively highlight image pixels around object edges, but the gradient image itself cannot well locate object edges. One tends to intuitively imagine object edges as thin lines or curve lines, yet each object edge corresponds to a thick highlighted zone or band in Fig. 2.14c.

As already explained, humans’ biological vision systems possess a mechanism that strengthens brightness or intensity contrast, so one may adopt a heuristic condition that “true” object edges abstracted in mathematical sense would cause locally-largest brightness contrast. Then to better locate object edges in the gradient image, one can take advantage of the spirit of *non-maximum suppression (NMS)*, which has already been presented in Sect. 2.2.4.

On the other hand, the non-maximum suppression method presented in Sect. 2.2.4 is not directly applicable here: It is apparently unreasonable to suppress pixels along directions of true object edges (as illustrated in Fig. 2.15a), be right along or obliquely along directions of true object edges.

In contrast, it is reasonable to suppress pixels only along normal directions of true object edges (as illustrated in Fig. 2.15b). Since the pixel values on each side of an object edge tends to or can fairly be regarded to be consistent locally in practical applications, object edges tend to form a set of virtual isohypses for the image pixel value function $I(u, v)$. Therefore, the normal direction of an object edge at a point can be treated as the point’s gradient direction.

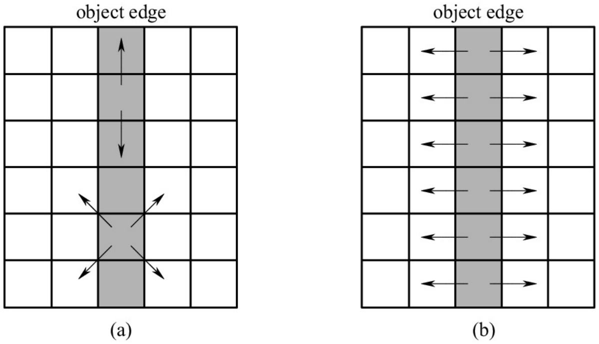


Fig. 2.15 NMS directions **a** unreasonable NMS directions; **b** reasonable NMS directions

Given a generic image point, although whether it is an edge point is unknown a priori, one can reasonably constrain the non-maximum suppression direction to its gradient direction (including the opposite gradient direction as well) in the gradient image. The point’s gradient direction defined on the continuous real bivariate domain $\boldsymbol{I}_{\mathrm{C}}$ in (2.3) is of infinite possibilities, depending on the u-part and v-part of the gradient. However, to be natural to discrete image processing, consideration is normally constrained to four potential non-maximum suppression directions, namely the vertical direction, the horizontal direction, the left-up-right-down direction, and the left-down-right-up direction, as illustrated in Fig. 2.16.

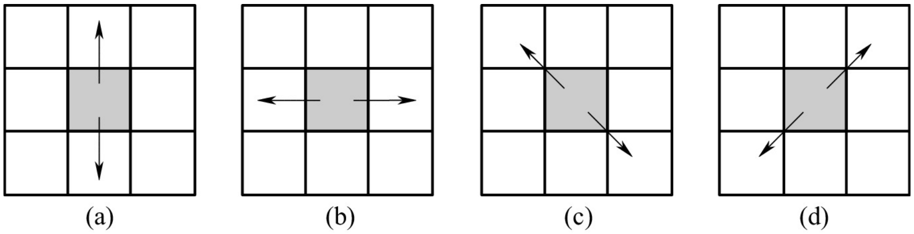

(a) (b) (c) (d)

Fig. 2.16 Directional NMS **a** vertical NMS; **b** horizontal NMS; **c** left-up-right-down NMS; **d** left-down-right-up NMS

To quickly determine a point's gradient direction namely its non-maximum suppression direction belongs to which category of the four potential directions illustrated in Fig. 2.16, one can compute absolute inner products between the point's gradient vector and the four normalized direction vectors $\boldsymbol{d}_1 = [01]^{\mathrm{T}}$, $\boldsymbol{d}_2 = [10]^{\mathrm{T}}$, $\boldsymbol{d}_3 = [\sqrt{1/2} - \sqrt{1/2}]^{\mathrm{T}}$, and $\boldsymbol{d}_4 = [\sqrt{1/2}\sqrt{1/2}]^{\mathrm{T}}$ respectively, and then find the direction vector associated with the largest absolute inner product:

$$\boldsymbol{d}_{\mathrm{G}}(u,v) = \mathrm{argmax}_{\boldsymbol{d}\in\{\boldsymbol{d}_{1\sim4}\}} \left| \boldsymbol{d}^{\mathrm{T}} \begin{bmatrix} I_u(u,v) \\ I_v(u,v) \end{bmatrix} \right|.$$

Once the point's discrete gradient vector is found, then the point in the gradient image is suppressed to zero if its gradient magnitude is smaller than at least one of its neighbour gradient magnitudes along the discrete gradient direction. Matlab code for directional non-maximum suppression is given as follows.

DirectionalNonMaximumSuppression.m

```
%% Directional Non-Maximum Suppression
function [Ig, Id] = DirectionalNonMaximumSuppression(Iu, Iv)
    H = size(Iu,1); W = size(Iu,2); Id = zeros(H,W);
    Ig = sqrt(Iu.^2+Iv.^2); IgD = zeros(H+2, W+2); IgD(2:H+1,2:W
    +1) = Ig;
    % Four discrete gradient and NMS directions
    dNMS = [[0;+1], ... % Vertical NMS direction
        [+1;0], ... % Horizontal NMS direction
        [+1;+1], ... % Left-down-right-up NMS direction
        [+1;-1]]; % Left-up-right-down NMS direction
    dF = [[0;1], ... % Normalized dNMS
        [1;0], [sqrt(1/2);sqrt(1/2)], [sqrt(1/2);-sqrt(1/2)]];
    for r = 1:H % v
        for c = 1:W % u
            % Inner products along four gradient directions
            dNm = abs([Iu(r,c), Iv(r,c)]*dF);
            dM = find(dNm == max(dNm)); dM = dM(1); Id(r,c) = dM
    ;
            uD = dNMS(1,dM); vD = dNMS(2,dM);
            % Directional NMS
            if (Ig(r,c)<IgD(r+1+vD,c+1+uD) || Ig(r,c)<IgD(r+1-vD
    ,c+1-uD))
                Ig(r,c) = 0;
            end
        end
    end
end
```

Directional non-maximum suppression is applied to the gradient image illustrated in Fig. 2.14c (or Fig. 2.17a), and the effect of directional non-maximum suppression is demonstrated in Fig. 2.17b. Compared with the gradient image before directional non-maximum suppression, the gradient image after directional non-maximum suppression becomes much more refined and can better locate object edges.

2.3.3 Double Thresholding

The refined gradient image such as illustrated in Fig. 2.17b well reflects potential locations of object edges—For a point in the refined gradient image, the larger its gradient magnitude is, the more likely it is an edge point—The next step is to determine which points of the refined gradient image can be treated as edge points, which normally involves *thresholding*.

A natural practice is single thresholding, namely using a single threshold to extract edge points: If a point’s gradient magnitude is larger than the threshold, then it is treated as an edge point; otherwise, it is not treated as an edge point. However, such practice tends to face a dilemma: If the threshold is set high, many true object edge segments would be missed. If the threshold is set low, many phantom edges would be extracted.

To overcome the dilemma faced by single thresholding, **double thresholding** can be adopted. The basic idea is to set two thresholds, namely a strong threshold and a weak threshold. The strong threshold aims at guaranteeing correctness of edge point extraction, whereas the weak threshold aims at not missing any potential edge point.

More specifically, for a point in the refined gradient image, if its gradient magnitude is larger than the strong threshold, then it is directly treated as an edge point and is called a *strong edge point* . If its gradient magnitude is not larger than the weak threshold, then it is directly treated as a non-edge point. If its gradient magnitude is larger than the weak threshold but not larger than the strong threshold, then it is treated as a potential edge point and is called a *weak edge point* .

Whether a weak edge point is finally treated as an edge point depends on whether it is on a weak edge point segment that connects to a strong edge point. In other words, if the weak edge point can connect to a strong edge point either directly or via other weak edge points, then it is finally treated as an edge point. Otherwise, namely if what the weak edge point can connect to are other weak edge points only, then it is finally treated as a non-edge point.

It is worth noting that above explanations are only for intuitive understanding. In practical applications, for implementation efficiency, people do not follow the logic of checking which weak edge points can connect to strong edge points, but actually follow the inverse logic of checking the strong edge points can connect to which weak edge points. This inverse logic can be conveniently realized by the *depth-first search*, i.e. the DFS method. Matlab code for double thresholding is given as follows.

DoubleThresholding.m

```
%% Double Thresholding
function imgE = DoubleThresholding(Ig, tS, tW)
    % tS: Strong threshold
    % tW: Weak threshold (by default 0.4 x strong threshold)
    if (nargin<2) tS = 13; tW = 0.4*tS;
    elseif (nargin<3) tW = 0.4*tS; end
    H = size(Ig,1)+2; W = size(Ig,2)+2;
    IgD = zeros(H, W); IgD(2:H-1,2:W-1) = Ig;
    imgS = zeros(H,W); imgW = zeros(H,W); imgE = zeros(H,W);
    iS = find(IgD>tS); imgS(iS) = 1; iW = find(IgD>tW); imgW(iW)
     = 1;
    % Edge point connection via DFS
    qDFS = zeros(1, length(iW)); % DFS queue
    k = length(iS); qDFS(1:k) = iS; imgW(iS) = 0;
    iOff = [-H-1, -H, -H+1, -1, +1, H-1, H, H+1]; %
    Neighbourhood offsets
    while(k>0) % DFS queue is not empty
        i = qDFS(k); imgE(i) = 1; k = k-1;
        for i2 = i + iOff
            if (1==imgW(i2)) k = k+1; qDFS(k) = i2; imgW(i2) =
    0; end
        end
    end
    imgE = imgE(2:H-1,2:W-1);
end
```

2.3.4 Canny Edge Detection

The consecutive procedures of Gaussian filtering, gradient image computation, directional non-maximum suppression, and double thresholding right form the famous **Canny edge detection** method [3]. Matlab code for the main stream of Canny edge detection is given as follows, where the involved sub-functions have already been given in previous sub-sections.

EdgeCanny.m

```
%% Canny Edge Detection
function [ptE, imgE] = EdgeCanny(img, tS, stdF, tW, wszF)
    % tS: Strong threshold
    % tW: Weak threshold (by default 0.4 x strong threshold)
    if (nargin<2) tS = 13; stdF = 1.4; tW = 0.4*tS; wszF = 2*
    ceil(stdF)+1;
    elseif (nargin<3) stdF = 1.4; tW = 0.4*tS; wszF = 2*ceil(
    stdF)+1;
    elseif (nargin<4) tW = 0.4*tS; wszF = 2*ceil(stdF)+1;
    elseif (nargin<5) wszF = 2*ceil(stdF)+1; end

    % Gaussian filtering over image
    imgDb = GaussianFilter2D(img, stdF, wszF);
    % Compute the gradient image % [Iu, Iv] = GradientImage(
    imgDb);
    [Iu, Iv] = GradientImageSobel(imgDb);
    % Non-maximum suppression along the gradient direction
    [Ig, Id] = DirectionalNonMaximumSuppression(Iu, Iv);
    % Double thresholding
    imgE = DoubleThresholding(Ig, tS, tW);
    [e_r, e_c] = find(imgE>0); ptE = [e_c’; e_r’];
end
```

Given the refined gradient image illustrated in Fig. 2.17b, the image of edge points obtained via strong thresholding and that obtained via weak thresholding are demonstrated in Fig. 2.18a , b respectively. The final image of Canny edge points obtained in the spirit of double thresholding is demonstrated in Fig. 2.18c.

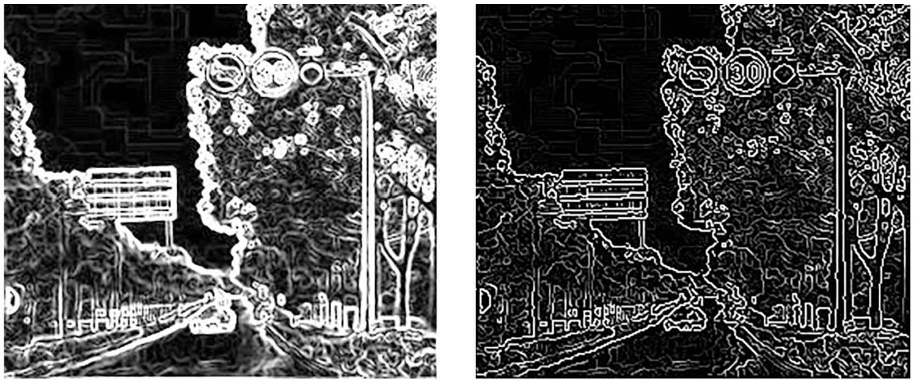
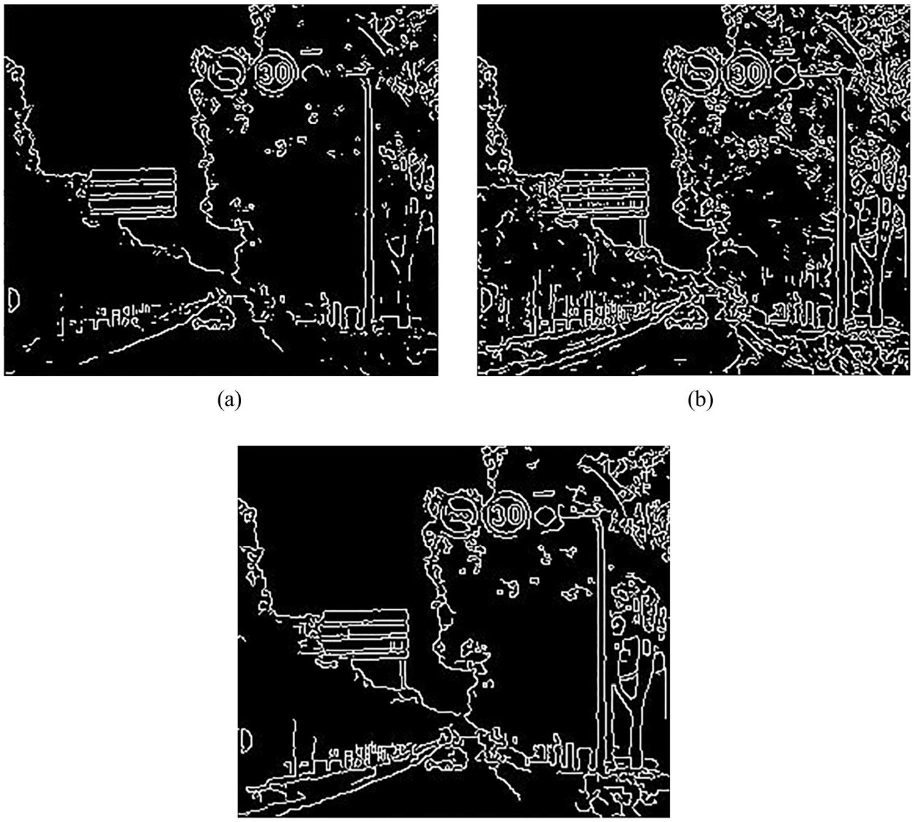

(a) (b)

Fig. 2.17 Directional non-maximum suppression **a** gradient image before directional NMS; **b** gradient image after directional NMS


(a) (b)

(c)

Fig. 2.18 Edge points **a** strong edge points; **b** weak edge points; **c** Canny edge points

As shown in Fig. 2.18a, the edge segments extracted by the strong threshold are rather broken and incomplete. As shown in Fig. 2.18b, the edge segments extracted by the weak threshold are suffused with small and tiny phantom edge segments. Neither of the two cases is desirable.

In contrast, as shown in Fig. 2.18c, the Canny edge points can cover visible object edges completely on one hand, and are exempt from small and tiny phantom edge segments on the other hands. This reflects well the advantage of double thresholding over single thresholding.

2.4 Corner Points

Besides object edges, corner points are another kind of salient geometric entities for visual analysis. A classical method of corner point detection is the **Harris corner detection** method [4], which consists of the following steps.

Step 1: Given an input image $\bar{I}(u, v)$ (namely the discrete subset of the image pixel value function $I(u, v)$), compute its gradient image

$$\nabla \bar{I}(u, v) \equiv \begin{bmatrix} \bar{I}_u(u, v) \\ \bar{I}_v(u, v) \end{bmatrix}$$

using methods presented in Sect. 2.3.1. $\bar{I}_u(u, v)$ and $\bar{I}_v(u, v)$ denote respectively the u-gradient image and the v-gradient image , which are the discrete subsets of the partial derivative functions $I_u(u, v)$ and $I_v(u, v)$ defined on $\boldsymbol{I}_{\mathrm{D}}$.

Step 2: Compute three auxiliary images $\bar{I}_{uu}(u, v)$, $\bar{I}_{vv}(u, v)$, and $\bar{I}_{uv}(u, v)$ as

$$\begin{aligned} \bar{I}_{uu}(u, v) = &\quad \bar{I}_u(u, v) \odot \bar{I}_u(u, v), \\ \bar{I}_{uv}(u, v) = &\quad \bar{I}_u(u, v) \odot \bar{I}_v(u, v), \\ \bar{I}_{vv}(u, v) = &\quad \bar{I}_v(u, v) \odot \bar{I}_v(u, v). \end{aligned}$$

Here, the author resorts to the *Hamadard product* $\odot$ to denote the element-wise product. In other words, the auxiliary image $\bar{I}_{uu}(u, v)$ is obtained via pixel-wise squaring of the u-gradient image $\bar{I}_u(u, v)$, the auxiliary image $\bar{I}_{vv}(u, v)$ is obtained via pixel-wise squaring of the v-gradient image $\bar{I}_v(u, v)$, and the auxiliary image $\bar{I}_{uv}(u, v)$ is obtained via pixel-wise product of the u-gradient image $\bar{I}_u(u, v)$ and the v-gradient image $\bar{I}_v(u, v)$.

Step 3: Perform Gaussian filtering over the three auxiliary images $\bar{I}_{uu}(u, v)$, $\bar{I}_{vv}(u, v)$, and $\bar{I}_{uv}(u, v)$ respectively. Denote their corresponding filtering results as $\hat{I}_{uu}(u, v)$, $\hat{I}_{vv}(u, v)$, and $\hat{I}_{uv}(u, v)$:

$$\begin{aligned} \hat{I}_{uu}(u, v) = &\quad g(u, v) * \bar{I}_{uu}(u, v), \\ \hat{I}_{uv}(u, v) = &\quad g(u, v) * \bar{I}_{uv}(u, v), \\ \hat{I}_{vv}(u, v) = &\quad g(u, v) * \bar{I}_{vv}(u, v), \end{aligned}$$

where $g(u, v)$ represents the Gaussian filter.

Step 4: Compute the **Harris image**

$$\begin{aligned} \bar{H}(u, v) = &\quad \hat{I}_{uu}(u, v) \odot \hat{I}_{vv}(u, v) - \hat{I}_{uv}(u, v) \odot \hat{I}_{uv}(u, v) \\ &\quad -k[\hat{I}_{uu}(u, v) + \hat{I}_{vv}(u, v)] \odot [\hat{I}_{uu}(u, v) + \hat{I}_{vv}(u, v)], \end{aligned} \tag{2.31}$$

where k is a parameter that is usually tuned between 0.04 and 0.15.

Step 5: Apply non-maximum suppression to the Harris image $\bar{H}(u,v)$ and then extract Harris corner points via thresholding—It is worth noting that one may treat the Harris image $\bar{H}(u,v)$ as the discrete subset of an imagined continuous bivariate function $H(u, v)$ and analyse $H(u, v)$ via quadratic interpolation presented in Sect. 2.1.3. This enables one to generate slightly refined estimates of corner locations.

Matlab code for the main stream of Harris corner detection is given as follows, where the involved sub-functions have already been given in previous sub-sections.

CornerHarris.m

```
%% Harris Corner Detection
function [ptH, imgH] = CornerHarris(img, tH, stdF, wszF, kH)
    if (nargin<2) tH = 5000; stdF = 1.4; wszF = 2*ceil(stdF)+1;
    kH = 0.06;
    elseif (nargin<3) stdF = 1.4; wszF = 2*ceil(stdF)+1; kH =
    0.06;
    elseif (nargin<4) wszF = 2*ceil(stdF)+1; kH = 0.06;
    elseif (nargin<5) kH = 0.06; end

    % Compute the gradient image and auxiliary images
    [Iu, Iv] = GradientImage(img);
    Iuv = Iu.*Iv; Iuu = Iu.^2; Ivv = Iv.^2;
    % Gaussian filtering over auxiliary images
    Iuu = GaussianFilter2D(Iuu, stdF, wszF);
    Ivv = GaussianFilter2D(Ivv, stdF, wszF);
    Iuv = GaussianFilter2D(Iuv, stdF, wszF);
    % Harris matrix
    detH = Iuu.*Ivv - Iuv.^2; trH = Iuu + Ivv;
    imgH = detH - kH*trH.^2; imgHOrg = imgH;
    % Non-maximum suppression and thresholding
    imgH = NonMaximumSuppression(imgH);
    [r, c] = find(imgH>tH); ptH = [c’; r’; imgH(r+(c-1)*size(
    imgH,1))’];
end
```

An example of Harris corner detection is demonstrated in Fig. 2.19, where the sub-figure(a) is the original image that shows regularly arranged squares printed on a panel. Figure 2.19b demonstrates results of Harris corner detection: the extracted corner points, which are marked by crosses, are intuitively consistent with object corners especially square corners.

2.5 Scale-Space Feature Extraction

A number of representative feature extraction methods have been presented in previous sub-sections. The kinds of feature points introduced by so far can be called **intuitive feature points**, as they are intuitive to humans’ biological vision systems. For example, what are detected by the Canny method as edge points can also be intuitively perceived by humans as edge points, and what are detected by the Harris method as corner points can also be intuitively perceived by humans as corner points. In fact, development of such intuitive feature extraction methods are well motivated by working mechanisms of humans’ biological vision systems.

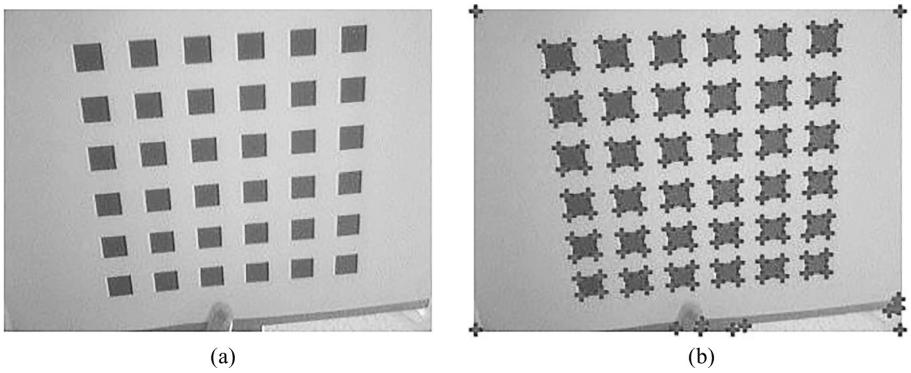

(a) (b)

Fig. 2.19 Corner points **a** original image; **b** Harris corner points

On the other hand, feature points useful in practical applications can also be **abstract feature points** that are not so intuitive to humans' biological vision systems. It is worth presenting a number of representative methods of abstract feature extraction, the first of which is deeply rooted in scale-space theory.

Raw images are inevitably with noises that can be caused by various factors. They are rarely used directly for image processing in practical applications and a preliminary step of image filtering is normally involved. For practice convenience and feasibility, instead of using certain theoretically better or even optimal filter (see the Wiener filter in Appendix C), one tends to adopt a "good-enough" **parametric filter** , the parameter(s) of which can be tuned empirically or adaptively. Besides, for the adopted parametric filter , its dominant parameter is normally the **filtering-scale** parameter that roughly determines the influence range of the filtering kernel . For simplicity yet without loss of generality, the author only considers the filtering-scale parameter here without entangling other *ad hoc* kinds of filter parameters.

When sufficient a priori knowledge is available in practical applications, the filtering-scale parameter may be tuned empirically or adjusted adaptively according to a priori knowledge. However, in practical applications where scenarios are rather general and a priori knowledge is lacking, there would be neither an acceptable way of tuning the filtering-scale parameter empirically nor an effective way of adjusting the filtering-scale parameter adaptively.

An advanced methodology of using the parametric filter is to circumvent tuning of the filtering-scale parameter but to analyse how a series of filtering-scale parameters influence the image and to extract image features that are salient and stable across filtered images associated with various filtering-scale parameters . An image processing module that takes advantage of the methodology has several inherent merits: First, it is exempt from explicit parameter tuning which is not easy in many practical applications. Second, it can possess a hierarchical perception ability which is rather similar to the multi-scale adaptive perception ability of humans' biological vision systems. Third, it has the potential to operate in rather general scenarios under various conditions. This advanced methodology is called **scale-space analysis**.

It seems that the spirit of scale-space analysis can be instantiated with whatever a parametric filter . However, right opposite to such intuition, if several fundamental criteria namely the causality, homogeneity, and isotropy criteria should be satisfied, the **parametric Gaussian filter** is the only possibility that can embody the spirit of scale-space analysis; see Appendix E for detailed clarification of this point.

2.5.1 Scale Space: Filtered Images Associated with Various Filtering-Scale Parameter

Given an image and the image pixel value function denoted as $I(u, v)$ ($1 \leqslant u \leqslant W$, $1 \leqslant v \leqslant H$), where W and H denote the image width and height respectively. The parametric Gaussian filter is

$$g(u, v, \sigma) = \frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}}. \tag{2.32}$$

Readers may notice the formalism difference between (2.32) and (E.7) in Appendix E. The latter is convenient to use in theoretical scale-space analysis, whereas the former is the conventionally adopted formalism for Gaussian filtering. Despite the formalism difference, (2.32) and (E.7) are essentially equivalent.

The filtered image function $I_{\mathrm{F}}(u, v, \sigma)$ defined over the trivariate domain $\{(u, v, \sigma)|\ 1 \leqslant u \leqslant W, 1 \leqslant v \leqslant H, \sigma \geqslant 0\}$, which represents the original image $I(u, v)$'s **scale space**, is obtained by convolving the parametric Gaussian filter (2.32) and the image pixel value function $I(u, v)$:

$$I_{\mathrm{F}}(u, v, \sigma) \equiv g(u, v, \sigma) * I(u, v) = \frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} * I(u, v). \tag{2.33}$$

Obviously, one cannot enumerate exhaustively all Gaussian filtered images of $I_{\mathrm{F}}(u, v, \sigma)$ in scale space by traversing the uncountable amount of filtering-scale parameters namely σ parameters across $[0, \infty)$. Instead, one enumerates Gaussian filtered images associated with a series of σ parameters to demonstrate how variation of the σ parameter causes variation of the Gaussian filtered image. An example set of Gaussian filtered images is illustrated in Fig. 2.20, whereas the sub-figure(a) illustrates an original image captured at Jungfraujoch in Suisse, and the eleven remaining sub-figures, displayed consecutively from left to right and from top to down, illustrate Gaussian filtered images associated with σ parameters $1.259\,9(\approx \sqrt[3]{2})$, $1.259\,9^2$, ..., $1.259\,9^{10}$, and $1.259\,9^{11}$ respectively.

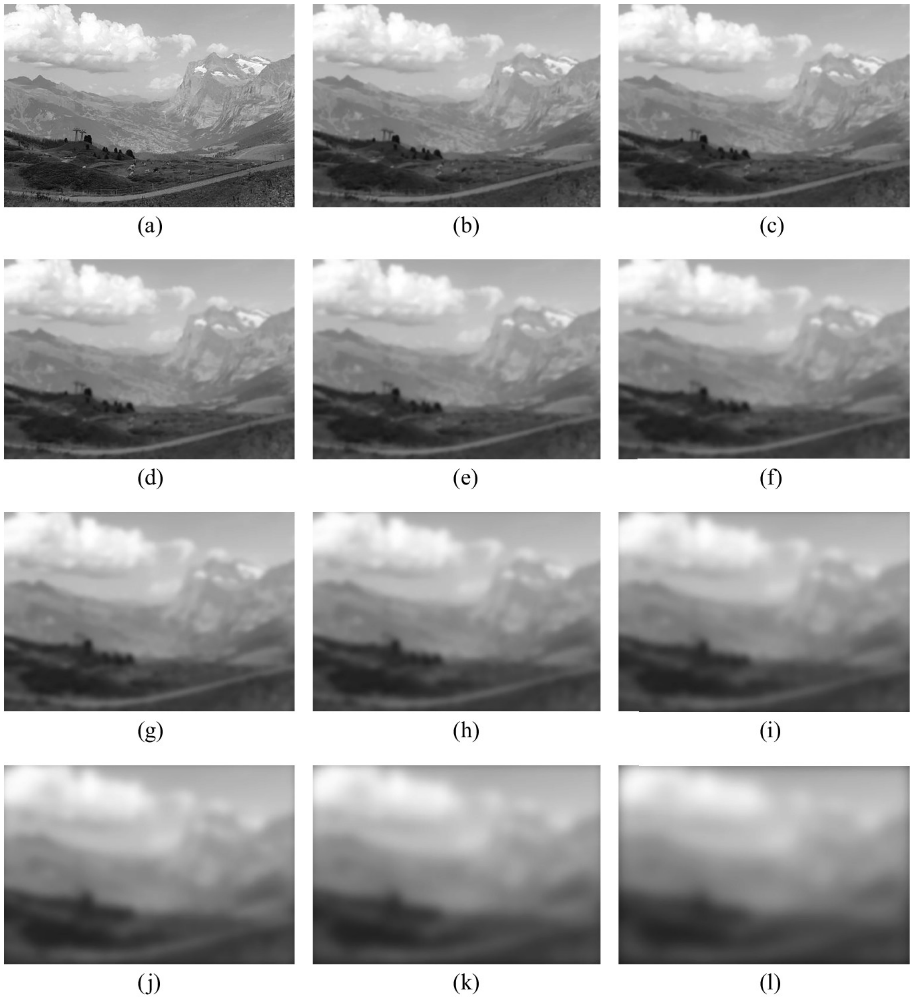


Fig. 2.20 Filtered images in scale space **a** original image; **b**, **c**, ..., **k**, **l** Gaussian filtered images associated with σ parameters $1.259\,9 (\approx \sqrt[3]{2})$, $1.259\,9^2$, ..., $1.259\,9^{10}$, and $1.259\,9^{11}$

In the example set of Gaussian filtered images , as the σ parameter namely the filtering-scale parameter of the parametric Gaussian filter increases, small areas' local patterns become less and less informative, whereas large areas' global patterns become more and more noticeable. In other words, as the filtering-scale parameter decreases, large areas' global patterns become less and less focused, whereas small areas' local patterns become more and more prominent. Intuitively speaking, no filtered image at a single filtering-scale can individually convey useful visual information at a wide range of scales. It is right the holistic combination of Gaussian filtered images in scale space that can comprehensively demonstrate salient features at all concerned scales.

2.5.2 Gaussian Pyramid

Instead of convolving the original image $I(u, v)$ directly with the parametric Gaussian filter at all concerned filtering-scales to form a discrete representation of scale space, Lowe propose a **pyramid representation** of Gaussian filtered images in scale space [5], which turns to be considerably more efficient than the direct discrete representation.

The basic spirit of pyramid representation consists in generating octaves of Gaussian filtered images . The initial image of the first octave is the original image and the first octave of Gaussian filtered images are generated by convolving its initial image respectively with a geometric series of σ parameters spanned between a prior smoothing σ parameter σ_0 and the prior smoothing double (namely $2\sigma_0$). The initial image of the second octave is down-sampled from the first octave’s last Gaussian filtered image by a factor of 2 and the second octave of Gaussian filtered images are generated by convolving its initial image respectively with the same geometric series of σ parameters spanned between σ_0 and $2\sigma_0$. Similarly, the initial image of the third octave is down-sampled from the second octave’s last Gaussian filtered image by a factor of 2 and the third octave of Gaussian filtered images are generated by convolving its initial image respectively with the same geometric series of σ parameters used previously for generation of the first and second octaves. The process continues like this until arriving at certain octave level.

An example of pyramid representation for the original image illustrated inFig. 2.20 is illustrated in Fig. 2.21, where for generation of each octave of Gaussian filtered images , the prior smoothing σ parameter is set as $\sigma_0 = 1.6$ and the geometric series of σ parameters is set as $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, 2\sigma_0\}$ with $\tau = \sqrt[3]{2} \approx 1.259\,9$.

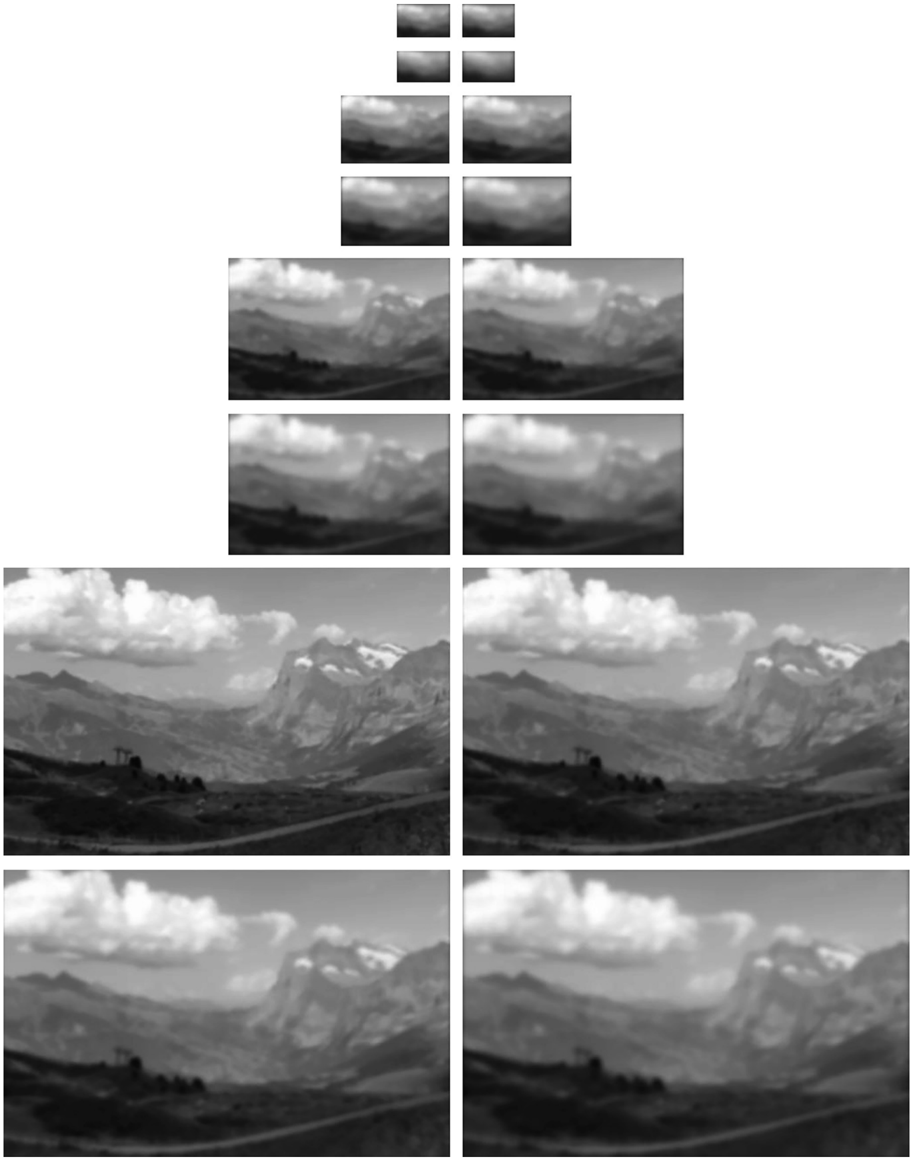


Fig. 2.21 Gaussian pyramid: pyramid representation of Gaussian filtered images in scale space (prior smoothing $\sigma_0 = 1.6$ and geometric series $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, 2\sigma_0\}$ with $\tau = \sqrt[3]{2}$)

It is worth noting that for each octave , it is the Gaussian filtered images associated with the last three σ parameters $\{\tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0\}$ (without σ_0) that actually represent the octave of scale space.

As the initial image of the n-th $(n \geqslant 2)$ octave is down-sampled from the $(n-1)$-th octave' image by a factor of 2, the Gaussian filtering effect corresponding to certain σ parameter in the n-th octave can be regarded as equivalent to that corresponding to 2σ from the $(n-1)$-th octave's perspective. Therefore, from the first octave's perspective, the first octave itself is represented directly by Gaussian filtered images associated with σ parameters $\{\tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0\}$, the second octave is represented equivalently by Gaussian filtered images associated with σ parameters $\{2\tau\sigma_0, 2\tau^2\sigma_0, 2\tau^3\sigma_0\}$, i.e. $\{\tau^4\sigma_0, \tau^5\sigma_0, \tau^6\sigma_0\}$, ..., the n-th octave is represented equivalently by Gaussian filtered images associated with σ parameters $\{2^{n-1}\tau\sigma_0, 2^{n-1}\tau^2\sigma_0, 2^{n-1}\tau^3\sigma_0\}$, i.e. $\{\tau^{3n-2}\sigma_0, \tau^{3n-1}\sigma_0, \tau^{3n}\sigma_0\}$ (note $\tau^3 = 2$).

The Gaussian pyramid that exploits down-sampling can equivalently represent Gaussian filtered images associated with σ parameters $\tau\sigma_0$, $\tau^2\sigma_0$, ..., $\tau^{3n-1}\sigma_0$, $\tau^{3n}\sigma_0$, $\tau^{3n+1}\sigma_0$, ... from the first octave 's perspective. In other words, the Gaussian pyramid strategy enables establishment of Gaussian filtered images , as if they are established directly from the original image according to the geometric series of σ parameters $\{\tau^k\sigma_0 | k = 1, 2, \ldots\}$.

Above is just an example, one can change τ to another value, say $\sqrt[4]{2}$. Then in this new case, each octave will involve five σ parameters $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0, \tau^4\sigma_0 = 2\sigma_0\}$ and it is the Gaussian filtered images associated with the last four σ parameters $\{\tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0, \tau^4\sigma_0\}$ that actually represent the octave of scale space.

Readers may wonder why the demonstrated Gaussian filtered images are always associated with a geometric series of σ parameters. In fact, Gaussian filtered images can be generated according to other kinds of σ parameters, yet usage of a geometric series of σ parameters facilitates image processing. Readers would naturally see the point after reading Sect. 2.5.4.

2.5.3 Scale-Invariant Laplacian

Like the gradient operator, the **Laplacian operator** $\Delta \equiv \frac{\partial^2}{\partial u^2} + \frac{\partial^2}{\partial v^2}$[10] is also a well-known image processing operator, dedicated to blob detection and blob scale fitting [6].

To facilitate understanding of the Laplacian operator's utility, consider an image with a disk blob of radius r_0 centred at (u_0, v_0); the image pixel value function is defined as

$$I_{\mathrm{D}}(u, v) = \begin{cases} I_0 > 0, & (u, v) \in D \\ 0, & \text{otherwise} \end{cases} \tag{2.34}$$

where the disk $D \equiv \{(u, v) | (u - u_0)^2 + (v - v_0)^2 \leqslant r_0^2\}$.

Convolve the image I_{D} and the parametric Gaussian filter (2.32):

$$I_{\mathrm{DF}}(u, v, \sigma) \equiv g(u, v, \sigma) * I_{\mathrm{D}}(u, v) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} * I_{\mathrm{D}}(u, v). \tag{2.35}$$

Perform the Laplacian operator on I_{DF}, which is equivalent to performing the Laplacian operator on the parametric Gaussian filter $g(u, v, \sigma)$ first and then convolving the image I_{D} and the Laplacian of $g(u, v, \sigma)$:

$$\Delta I_{\mathrm{DF}}(u, v, \sigma) = \Delta g(u, v, \sigma) * I_{\mathrm{D}}(u, v). \tag{2.36}$$

Readers may verify this useful property in two ways: either via analysis directly based on convolution definition, or by exploiting the relationship between the derivative operator and the Laplace or Fourier transform . Details are left to readers for practice.

Compute the Laplacian of the parametric Gaussian filter:

$$\begin{aligned} \frac{\partial^2 g}{\partial u^2} &= \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{u^2}{\sigma^6} - \frac{1}{\sigma^4} \right), \quad \frac{\partial^2 g}{\partial v^2} = \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{v^2}{\sigma^6} - \frac{1}{\sigma^4} \right), \\ \Delta g &= \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{u^2+v^2}{\sigma^6} - \frac{2}{\sigma^4} \right). \end{aligned} \tag{2.37}$$

Substitute (2.34) and (2.37) into (2.36) and obtain

$$\begin{aligned} & \Delta I_{\mathrm{DF}}(u, v, \sigma) \\ = & \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{u^2+v^2}{\sigma^6} - \frac{2}{\sigma^4} \right) * I_{\mathrm{D}}(u, v) \\ = & \frac{I_0}{2\pi} \iint_D \mathrm{e}^{-\frac{(\delta u - u' + u_0)^2 + (\delta v - v' + v_0)^2}{2\sigma^2}} \left(\frac{(\delta u - u' + u_0)^2 + (\delta v - v' + v_0)^2}{\sigma^6} - \frac{2}{\sigma^4} \right) \mathrm{d}u' \mathrm{d}v' \\ = & \frac{I_0}{2\pi} \iint_{D_0} \mathrm{e}^{-\frac{(\delta u - u')^2 + (\delta v - v')^2}{2\sigma^2}} \left(\frac{(\delta u - u')^2 + (\delta v - v')^2}{\sigma^6} - \frac{2}{\sigma^4} \right) \mathrm{d}u' \mathrm{d}v', \end{aligned} \tag{2.38}$$

where $\delta u \equiv u - u_0$, $\delta v \equiv v - v_0$, and the disk $D_0 \equiv \{(u, v) | u^2 + v^2 \leqslant r_0^2\}$.

Consider the Laplacian of I_{DF} about the disk blob center (u_0, v_0), namely ΔI_{DF} in (u_0, v_0)'s neighbourhood $\{(u, v) | u - u_0 \approx 0, v - v_0 \approx 0\}$. Denote $\delta r^2 \equiv \delta u^2 + \delta v^2$ and $[\delta u, \delta v] \equiv \delta r[\cos \phi, \sin \phi]$. Perform the integral variable transform $(u', v') = (r \cos \theta + \delta u, r \sin \theta + \delta v)$ and obtain[11]

$$\Delta I_{\mathrm{DF}}(u, v, \sigma) = \frac{I_0}{2\pi} \int_0^{2\pi} \int_0^{r(\theta)} \mathrm{e}^{-\frac{r^2}{2\sigma^2}} \left(\frac{r^2}{\sigma^6} - \frac{2}{\sigma^4} \right) r \mathrm{d}r \mathrm{d}\theta,$$

where

$$\begin{aligned} r(\theta) &= \sqrt{r_0^2 - (\delta u \sin \theta - \delta v \cos \theta)^2} - (\delta u \cos \theta + \delta v \sin \theta) \\ &= \sqrt{r_0^2 - \delta r^2 \sin^2 (\theta - \phi)} - \delta r \cos (\theta - \phi). \end{aligned} \tag{2.39}$$

Further derive $\Delta I_{\mathrm{DF}}(u, v, \sigma)$ as

$$\begin{aligned} \Delta I_{\mathrm{DF}}(u, v, \sigma) = & \frac{I_0}{\pi \sigma^2} \int_0^{2\pi} \int_0^{r(\theta)} \mathrm{e}^{-\frac{r^2}{2\sigma^2}} \left(\frac{r^2}{2\sigma^2} - 1 \right) \mathrm{d} \frac{r^2}{2\sigma^2} \mathrm{d}\theta \\ = & \frac{I_0}{\pi \sigma^2} \int_0^{2\pi} \int_0^{\frac{r(\theta)^2}{2\sigma^2}} \mathrm{e}^{-x} (x - 1) \mathrm{d}x \mathrm{d}\theta \quad \left(\text{denote } x \equiv \frac{r^2}{2\sigma^2} \right) \\ = & \frac{I_0}{\pi \sigma^2} \int_0^{2\pi} (-x \mathrm{e}^{-x} |_0^{\frac{r(\theta)^2}{2\sigma^2}}) \mathrm{d}\theta = -\frac{I_0}{2\pi \sigma^4} \int_0^{2\pi} r(\theta)^2 \mathrm{e}^{-\frac{r(\theta)^2}{2\sigma^2}} \mathrm{d}\theta. \end{aligned} \tag{2.40}$$

Note that $r(\theta)$ is a periodic function in terms of θ with the period 2π. Substitute (2.39) into (2.40) and obtain

$$\begin{aligned}
&\Delta I_{\mathrm{DF}}(u,v,\sigma)\\
=\;& -\frac{I_0}{2\pi\sigma^4}\int_0^{2\pi} r(\theta)^2 \mathrm{e}^{-\frac{r(\theta)^2}{2\sigma^2}}\mathrm{d}\theta\\
=\;& -\frac{I_0}{2\pi\sigma^4}\int_0^{2\pi} r(\theta+\phi)^2 \mathrm{e}^{-\frac{r(\theta+\phi)^2}{2\sigma^2}}\mathrm{d}\theta\\
=\;& -\frac{I_0}{2\pi\sigma^4}\int_0^{2\pi} \left(\sqrt{r_0^2-\delta r^2\sin^2\theta}-\delta r\cos\theta\right)^2 \mathrm{e}^{-\frac{\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}-\delta r\cos\theta\right)^2}{2\sigma^2}}\mathrm{d}\theta\\
=\;& \frac{I_0}{\pi\sigma^2}\int_0^{2\pi}\left[\sum_{k=0}^{\infty}\frac{1}{k!}\left(-\frac{\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}-\delta r\cos\theta\right)^2}{2\sigma^2}\right)^{k+1}\right]\mathrm{d}\theta\\
=\;& \frac{I_0}{\pi\sigma^2}\sum_{k=0}^{\infty}\frac{1}{k!}\left(-\frac{1}{2\sigma^2}\right)^{k+1}\left[\int_0^{2\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}-\delta r\cos\theta\right)^{2k+2}\mathrm{d}\theta\right].
\end{aligned}\tag{2.41}$$

Analyse the term in the bracket in (2.41)

$$\begin{aligned}
&\int_0^{2\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}-\delta r\cos\theta\right)^{2k+2}\mathrm{d}\theta\\
=\;& \sum_{i=0}^{2k+2} C_{2k+2}^{i}\left[\int_0^{2\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}\right)^{2k+2-i}(-\delta r\cos\theta)^i\mathrm{d}\theta\right],
\end{aligned}$$

in which if i is odd, then

$$\begin{aligned}
&\int_0^{2\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}\right)^{2k+2-i}(-\delta r\cos\theta)^i\mathrm{d}\theta\\
=\;& -\int_0^{\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}\right)^{2k+2-i}(\delta r\cos\theta)^i\mathrm{d}\theta\\
& -\int_0^{\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2(\theta+\pi)}\right)^{2k+2-i}(\delta r\cos(\theta+\pi))^i\mathrm{d}\theta\\
=\;& -\int_0^{\pi}\left(\sqrt{r_0^2-\delta r^2\sin^2\theta}\right)^{2k+2-i}[(\delta r\cos\theta)^i+(-\delta r\cos\theta)^i]\mathrm{d}\theta=0.
\end{aligned}$$

So

$$\begin{aligned}
&\int\limits_0^{2\pi} \left(\sqrt{r_0^2 - \delta r^2 \sin^2\theta} - \delta r \cos\theta\right)^{2k+2} \mathrm{d}\theta \\
= \quad & \sum\nolimits_{i=0}^{k+1} C_{2k+2}^{2i} \left[\int\limits_0^{2\pi} \left(\sqrt{r_0^2 - \delta r^2 \sin^2\theta}\right)^{2k+2-2i} (-\delta r \cos\theta)^{2i} \mathrm{d}\theta\right] \\
= \quad & \sum\nolimits_{i=0}^{k+1} C_{2k+2}^{2i} \left[\int\limits_0^{2\pi} \left(r_0^2 - \delta r^2 \sin^2\theta\right)^{k+1-i} \left(\delta r^2 \cos^2\theta\right)^{i} \mathrm{d}\theta\right].
\end{aligned} \tag{2.42}$$

For infinitesimal analysis of the Laplacian ΔI_{DF} about the disk blob center (u_0, v_0), consider terms of δr only up to four orders [12] in (2.41). For the purpose, first consider terms of δr only up to four orders in (2.42)

$$\begin{aligned}
&\int\limits_0^{2\pi} \left(\sqrt{r_0^2 - \delta r^2 \sin^2\theta} - \delta r \cos\theta\right)^{2k+2} \mathrm{d}\theta \\
= \quad & \sum\nolimits_{i=0}^{2} C_{2k+2}^{2i} \left[\int\limits_0^{2\pi} \left(r_0^2 - \delta r^2 \sin^2\theta\right)^{k+1-i} \left(\delta r^2 \cos^2\theta\right)^{i} \mathrm{d}\theta\right] + o(\delta r^4) \\
= \quad & \int\limits_0^{2\pi} \{[r_0^{2k+2} - C_{k+1}^1 r_0^{2k} \delta r^2 \sin^2\theta + C_{k+1}^2 r_0^{2k-2} \delta r^4 \sin^4\theta + o(\delta r^4)] \\
& + C_{2k+2}^2 [r_0^{2k} \delta r^2 \cos^2\theta - C_k^1 r_0^{2k-2} \delta r^4 \sin^2\theta \cos^2\theta + o(\delta r^4)] \\
& + C_{2k+2}^4 [r_0^{2k-2} \delta r^4 \cos^4\theta + o(\delta r^4)]\} \mathrm{d}\theta + o(\delta r^4) \\
& (\text{omit orders higher than } 4) \\
\approx \quad & 2\pi r_0^{2k+2} + (C_{2k+2}^2 - C_{k+1}^1) \pi r_0^{2k} \delta r^2 + \left(\tfrac{3}{4} C_{2k+2}^4 + \tfrac{3}{4} C_{k+1}^2 - \tfrac{1}{4} C_{2k+2}^2 C_k^1\right) \pi r_0^{2k-2} \delta r^4 \\
= \quad & 2\pi r_0^{2k+2} + 2k(k+1) \pi r_0^{2k} \delta r^2 + \tfrac{k^2(k^2-1)}{2} \pi r_0^{2k-2} \delta r^4.
\end{aligned} \tag{2.43}$$

For derivation of (2.43), the following integrals are used.

$$\begin{aligned}
\int\limits_0^{2\pi} \cos^2\theta \mathrm{d}\theta = \int\limits_0^{2\pi} \sin^2\theta \mathrm{d}\theta = & \quad \pi, \\
\int\limits_0^{2\pi} \cos^2\theta \sin^2\theta \mathrm{d}\theta = & \quad \tfrac{\pi}{4}, \\
\int\limits_0^{2\pi} \cos^4\theta \mathrm{d}\theta = \int\limits_0^{2\pi} \sin^4\theta \mathrm{d}\theta = & \quad \tfrac{3\pi}{4}.
\end{aligned}$$

Substitute (2.43) into (2.41) and obtain

$$
\begin{aligned}
& \Delta I_{\mathrm{DF}}(u,v,\sigma) \\
\approx\; & \frac{I_0}{\sigma^2}\sum_{k=0}^{\infty}\frac{1}{k!}\left(-\frac{1}{2\sigma^2}\right)^{k+1}\left[2r_0^{2k+2}+2k(k+1)r_0^{2k}\delta r^2+\frac{k^2(k^2-1)}{2}r_0^{2k-2}\delta r^4\right] \\
=\; & \frac{I_0}{\sigma^2}\left[2\sum_{k=0}^{\infty}\frac{1}{k!}\left(-\frac{r_0^2}{2\sigma^2}\right)^{k+1}-\frac{\delta r^2}{\sigma^2}\sum_{k=1}^{\infty}\frac{k+1}{(k-1)!}\left(-\frac{r_0^2}{2\sigma^2}\right)^{k}+\right. \\
& \left.\frac{\delta r^4}{8\sigma^4}\sum_{k=2}^{\infty}\frac{(k+1)k}{(k-2)!}\left(-\frac{r_0^2}{2\sigma^2}\right)^{k-1}\right] \\
=\; & \frac{I_0}{\sigma^2}\left[2\sum_{k=0}^{\infty}\frac{1}{k!}x^{k+1}-\frac{\delta r^2}{\sigma^2}\sum_{k=1}^{\infty}\frac{k+1}{(k-1)!}x^{k}+\frac{\delta r^4}{8\sigma^4}\sum_{k=2}^{\infty}\frac{(k+1)k}{(k-2)!}x^{k-1}\right] \quad \left(x\equiv-\frac{r_0^2}{2\sigma^2}\right) \\
=\; & \frac{I_0}{\sigma^2}\left[2\sum_{k=0}^{\infty}\frac{x^{k+1}}{k!}-\frac{\delta r^2}{\sigma^2}\left(\sum_{k=1}^{\infty}\frac{x^{k+1}}{(k-1)!}\right)'+\frac{\delta r^4}{8\sigma^4}\left(\sum_{k=2}^{\infty}\frac{x^{k+1}}{(k-2)!}\right)''\right] \\
=\; & \frac{I_0}{\sigma^2}\left[2x\mathrm{e}^{x}-\frac{\delta r^2}{\sigma^2}\left(x^2\mathrm{e}^{x}\right)'+\frac{\delta r^4}{8\sigma^4}\left(x^3\mathrm{e}^{x}\right)''\right] \\
=\; & \frac{I_0}{\sigma^2}x\mathrm{e}^{x}\left[2-\frac{\delta r^2}{\sigma^2}(x+2)+\frac{\delta r^4}{8\sigma^4}(x^2+6x+6)\right] \\
=\; & \frac{I_0}{\sigma^2}\left(-\frac{r_0^2}{2\sigma^2}\right)\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left[2-\frac{\delta r^2}{\sigma^2}\left(2-\frac{r_0^2}{2\sigma^2}\right)+\frac{\delta r^4}{8\sigma^4}\left(\left(-\frac{r_0^2}{2\sigma^2}\right)^2+6\left(-\frac{r_0^2}{2\sigma^2}\right)+6\right)\right] \\
=\; & -\frac{I_0r_0^2}{2\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left[2-\left(\frac{2}{\sigma^2}-\frac{r_0^2}{2\sigma^4}\right)\delta r^2+\left(\frac{r_0^4}{32\sigma^8}-\frac{3r_0^2}{8\sigma^6}+\frac{3}{4\sigma^4}\right)\delta r^4\right] \\
=\; & \frac{I_0r_0^2}{\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left[-1+\left(\frac{1}{\sigma^2}-\frac{r_0^2}{4\sigma^4}\right)(\delta u^2+\delta v^2)-\left(\frac{r_0^4}{64\sigma^8}-\frac{3r_0^2}{16\sigma^6}+\frac{3}{8\sigma^4}\right)(\delta u^2+\delta v^2)^2\right].
\end{aligned}
\tag{2.44}
$$

The derived result described in (2.44) reflects a number of interesting properties of $\Delta I_{\mathrm{DF}}(u,v,\sigma)$:

- No matter for which σ parameter, the disk blob center (u_0,v_0) is always an extremum of the bivariate function $\Delta I_{\mathrm{DF}}(u,v,\sigma)$ in terms of $(u,\, v)$. To verify this property, note

  $$
  \begin{aligned}
  \frac{\partial}{\partial u}\Delta I_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0} &= \frac{2I_0r_0^2}{\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left(\frac{1}{\sigma^2}-\frac{r_0^2}{4\sigma^4}\right)\delta u\big|_{u=u_0,v=v_0}=0, \\
  \frac{\partial}{\partial v}\Delta I_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0} &= \frac{2I_0r_0^2}{\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left(\frac{1}{\sigma^2}-\frac{r_0^2}{4\sigma^4}\right)\delta v\big|_{u=u_0,v=v_0}=0, \\
  \nabla^2\Delta I_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0} &= \frac{2I_0r_0^2}{\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left(\frac{1}{\sigma^2}-\frac{r_0^2}{4\sigma^4}\right)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},
  \end{aligned}
  $$

  the last of which, namely the Hessian matrix, is either positive definite or negative definite. When $\sigma<\frac{r_0}{2}$, the Hessian matrix is negative definite. When $\sigma>\frac{r_0}{2}$, the Hessian matrix is positive definite. When $\sigma=\frac{r_0}{2}$, the Hessian matrix is degenerate.
- The critical value $\sigma=\frac{r_0}{2}$ corresponds to the extremum (more specifically, the minimum) of the univariate function $\Delta I_{\mathrm{DF}}(u_0,v_0,\sigma)$ in terms of σ. To verify this property, note

$$\frac{\partial}{\partial\sigma}\Delta I_{\mathrm{DF}}(u_0, v_0, \sigma) = \frac{I_0 r_0^2}{\sigma^7}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}(4\sigma^2 - r_0^2) = 0 \Longleftrightarrow \sigma_c = \frac{r_0}{2}.$$

When $\sigma < \frac{r_0}{2}$, the derivative $\frac{\partial}{\partial\sigma}\Delta I_{\mathrm{DF}}(u_0, v_0, \sigma) < 0$. When $\sigma > \frac{r_0}{2}$, the derivative $\frac{\partial}{\partial\sigma}\Delta I_{\mathrm{DF}}(u_0, v_0, \sigma) > 0$.

- The point $\left(u_0, v_0, \frac{r_0}{2}\right)$ corresponds to the extremum (more specifically, the minimum) of the trivariate function $\Delta I_{\mathrm{DF}}(u, v, \sigma)$. To verify this property, first note from above two properties that the first-order partial derivatives of $\Delta I_{\mathrm{DF}}(u, v, \sigma)$ at $\left(u_0, v_0, \frac{r_0}{2}\right)$ are all zero. Second, note from above firstproperty

$$\begin{aligned} &\frac{\partial^2}{\partial u \partial\sigma}\Delta I_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0, \sigma=\frac{r_0}{2}} \propto \delta u\big|_{u=u_0} = 0, \\ &\frac{\partial^2}{\partial v \partial\sigma}\Delta I_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0, \sigma=\frac{r_0}{2}} \propto \delta v\big|_{v=v_0} = 0, \\ &\begin{bmatrix} \frac{\partial^2}{\partial u^2} & \frac{\partial^2}{\partial u \partial v} \\ [9pt]\frac{\partial^2}{\partial u \partial v} & \frac{\partial^2}{\partial v^2} \end{bmatrix}\Delta I_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0, \sigma=\frac{r_0}{2}} = \mathbf{0}. \end{aligned}$$

Third, note from above second property

$$\Delta I_{\mathrm{DF}\sigma\sigma}\left(u_0, v_0, \frac{r_0}{2}\right) \equiv \frac{\partial^2}{\partial\sigma^2}\Delta I_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0, \sigma=\frac{r_0}{2}} = \frac{512 I_0}{r_0^4}\mathrm{e}^{-2} > 0.$$

So according to (2.44), in $\left(u_0, v_0, \frac{r_0}{2}\right)$'s infinitesimal neighbourhood

$$\begin{aligned} \Delta I_{\mathrm{DF}}(u, v, \sigma) \approx\ & \Delta I_{\mathrm{DF}}\left(u_0, v_0, \frac{r_0}{2}\right) + \frac{1}{2}\Delta I_{\mathrm{DF}\sigma\sigma}\left(u_0, v_0, \frac{r_0}{2}\right)\delta\sigma^2 \\ & - \left[\frac{I_0 r_0^2}{\sigma^4}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}\left(\frac{r_0^4}{64\sigma^8} - \frac{3r_0^2}{16\sigma^6} + \frac{3}{8\sigma^4}\right)\bigg|_{\sigma=\frac{r_0}{2}}\right](\delta u^2 + \delta v^2)^2 \\ =\ & -\frac{16\mathrm{e}^{-2} I_0}{r_0^2} + \frac{256\mathrm{e}^{-2} I_0}{r_0^4}\delta\sigma^2 + \frac{32\mathrm{e}^{-2} I_0}{r_0^6}(\delta u^2 + \delta v^2)^2 \\ \geqslant\ & -\frac{16\mathrm{e}^{-2} I_0}{r_0^2} = \Delta I_{\mathrm{DF}}\left(u_0, v_0, \frac{r_0}{2}\right) \end{aligned}$$

which implies this claimed property that $\left(u_0, v_0, \frac{r_0}{2}\right)$ corresponds to the extremum (more specifically, the minimum) of $\Delta I_{\mathrm{DF}}(u, v, \sigma)$. Some extra explanations hover over above infinitesimal analysis: The Hessian matrix elements associated with u and v are degenerate, so δu and δv terms of orders higher than two need to be analysed. In contrast, the unique non-degenerate Hessian matrix element is the one purely associated with σ, so the second-order term of $\delta\sigma$ is still considered.

These properties convey that one can take advantage of the Laplacian operator to not only localize the disk blob's position but also identify its size, by finding the extremum of the Laplacian of the filtered image function $I_{\mathrm{DF}}(u, v, \sigma)$. Although the principle is derived above via analysis of the disk blob model, the principle is generally applicable to detection of salient blobs (or blob-like areas) in images.

More specifically, establish the original image's scale space that is generically represented by the filtered image function $I_{\mathrm{F}}(u, v, \sigma)$. Compute its Laplacian $\Delta I_{\mathrm{F}}(u, v, \sigma)$. Find extrema of the Laplacian $\Delta I_{\mathrm{F}}(u, v, \sigma)$, which in practice is normally realized by non-maximum suppression throughout discrete scale space .

The Laplacian operator's utility has been clarified, yet direct usage of the Laplacian operator has a limitation: it is not scale-invariant. To see what the limitation actually means, pay attention to the extremum value of $\Delta I_{\mathrm{DF}}(u, v, \sigma)$:

$$\Delta I_{\mathrm{DFm}} \equiv \Delta I_{\mathrm{DF}}\left(u_0, v_0, \frac{r_0}{2}\right) = -\frac{16\mathrm{e}^{-2} I_0}{r_0^2}. \tag{2.45}$$

The extremum value of $\Delta I_{\mathrm{DF}}(u, v, \sigma)$ depends on the blob scale: the larger the blob scale is, the smaller the absolute value of ΔI_{DFm} is; the smaller the blob scale is, the larger the absolute value of ΔI_{DFm} is. In other words, smaller blobs have more prominent Laplacian extrema in scale space than larger blobs, which in practical applications tends to bias detection output towards smaller blobs while suppressing larger blobs.

To overcome the limitation, instead of using the Laplacian operator directly, one can take advantage of the **scale-invariant Laplacian operator** that is defined as $\sigma^2 \Delta$, namely the Laplacian operator multiplied by the square of the σ parameter. Denote

$$\hat{I}_{\mathrm{DF}}(u, v, \sigma) \equiv \sigma^2\, g(u, v, \sigma) * I_{\mathrm{D}}(u, v) = \sigma^2 I_{\mathrm{DF}}(u, v, \sigma). \tag{2.46}$$

Consider the Laplacian of $\hat{I}_{\mathrm{DF}}$ still in the disk blob center (u_0, v_0)'s neighbourhood. According to (2.44)

$$\begin{aligned} \Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma) = &\quad \sigma^2 \Delta I_{\mathrm{DF}}(u, v, \sigma) \\ = &\quad \frac{I_0 r_0^2}{\sigma^2} \mathrm{e}^{-\frac{r_0^2}{2\sigma^2}} \left[-1 + \left(\frac{1}{\sigma^2} - \frac{r_0^2}{4\sigma^4}\right)(\delta u^2 + \delta v^2)\right. \\ &\quad \left. - \left(\frac{r_0^4}{64\sigma^8} - \frac{3r_0^2}{16\sigma^6} + \frac{3}{8\sigma^4}\right)(\delta u^2 + \delta v^2)^2\right]. \end{aligned} \tag{2.47}$$

The derived result in (2.47) reflects following properties of $\Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma)$, the first three of which are similar to above properties of $\Delta I_{\mathrm{DF}}(u, v, \sigma)$ reflected by (2.44):

- No matter for which σ parameter, the disk blob center (u_0, v_0) is always an extremum of the bivariate function $\Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma)$ in terms of (u, v). To verify this property, note

$$\begin{aligned} \frac{\partial}{\partial u} \Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0} &= \frac{2I_0 r_0^2}{\sigma^2} \mathrm{e}^{-\frac{r_0^2}{2\sigma^2}} \left(\frac{1}{\sigma^2} - \frac{r_0^2}{4\sigma^4}\right) \delta u\big|_{u=u_0, v=v_0} = 0, \\ \frac{\partial}{\partial v} \Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0} &= \frac{2I_0 r_0^2}{\sigma^2} \mathrm{e}^{-\frac{r_0^2}{2\sigma^2}} \left(\frac{1}{\sigma^2} - \frac{r_0^2}{4\sigma^4}\right) \delta v\big|_{u=u_0, v=v_0} = 0, \\ \nabla^2 \Delta \hat{I}_{\mathrm{DF}}(u, v, \sigma)\big|_{u=u_0, v=v_0} &= \frac{2I_0 r_0^2}{\sigma^2} \mathrm{e}^{-\frac{r_0^2}{2\sigma^2}} \left(\frac{1}{\sigma^2} - \frac{r_0^2}{4\sigma^4}\right) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \end{aligned}$$

  the last of which, namely the Hessian matrix, is positive definite when $\sigma > \frac{r_0}{2}$.
- The critical value $\sigma = \frac{\sqrt{2} r_0}{2}$ corresponds to the extremum (more specifically, the minimum) of the univariate function $\Delta \hat{I}_{\mathrm{DF}}(u_0, v_0, \sigma)$ in terms of σ. To verify this

property, note

$$\frac{\partial}{\partial\sigma}\Delta\hat{I}_{\mathrm{DF}}(u_0,v_0,\sigma) = \frac{I_0 r_0^2}{\sigma^5}\mathrm{e}^{-\frac{r_0^2}{2\sigma^2}}(2\sigma^2 - r_0^2) = 0 \Longleftrightarrow \sigma_c = \frac{\sqrt{2}r_0}{2}.$$

- The point $\left(u_0, v_0, \frac{\sqrt{2}r_0}{2}\right)$ corresponds to the extremum (more specifically, the minimum) of the trivariate function $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$. To verify this property, first note the first-order partial derivatives of $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$ at $\left(u_0, v_0, \frac{\sqrt{2}r_0}{2}\right)$ are all zero. Second, note

$$\begin{aligned} \frac{\partial^2}{\partial u\partial\sigma}\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0,\sigma=\frac{\sqrt{2}r_0}{2}} &\propto \delta u\big|_{u=u_0} = 0, \\ \frac{\partial^2}{\partial v\partial\sigma}\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0,\sigma=\frac{\sqrt{2}r_0}{2}} &\propto \delta v\big|_{v=v_0} = 0, \\ \begin{bmatrix} \frac{\partial^2}{\partial u^2} & \frac{\partial^2}{\partial u\partial v} \\ {[6pt]}\,\frac{\partial^2}{\partial u\partial v} & \frac{\partial^2}{\partial v^2} \end{bmatrix}\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0,\sigma=\frac{\sqrt{2}r_0}{2}} &= \frac{4I_0\mathrm{e}^{-1}}{r_0^2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} > 0. \end{aligned}$$

  Third, note

$$\frac{\partial^2}{\partial\sigma^2}\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0,\sigma=\frac{\sqrt{2}r_0}{2}} = \frac{16I_0}{r_0^2}\mathrm{e}^{-1} > 0.$$

  So

$$\nabla^2\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)\big|_{u=u_0,v=v_0,\sigma=\frac{\sqrt{2}r_0}{2}} = \begin{bmatrix} \frac{4I_0\mathrm{e}^{-1}}{r_0^2} & & \\ & \frac{4I_0\mathrm{e}^{-1}}{r_0^2} & \\ & & \frac{16I_0\mathrm{e}^{-1}}{r_0^2} \end{bmatrix} > 0.$$

  In other words, the Hessian matrix of the trivariate function $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$ is positive definite. Therefore, $\left(u_0, v_0, \frac{\sqrt{2}r_0}{2}\right)$ corresponds to the extremum (more specifically, the minimum) of $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$.

- The extremum value of $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$ is scale-invariant. To verify this property, compute the extremum value of $\Delta\hat{I}_{\mathrm{DF}}(u,v,\sigma)$:

$$\Delta\hat{I}_{\mathrm{DFm}} \equiv \Delta\hat{I}_{\mathrm{DF}}\left(u_0, v_0, \frac{\sqrt{2}r_0}{2}\right) = -2I_0\mathrm{e}^{-1}, \tag{2.48}$$

  which does not depend on the blob scale. In other words, larger blobs have as same prominent Laplacian extrema in scale space as smaller blobs.

The first three of above properties imply that like the Laplacian operator Δ, the scale-invariant Laplacian operator $\sigma^2\Delta$ also enables localization of disk blobs' positions and identification of their sizes, which is achieved by finding extrema of the scale-invariant Laplacian of the filtered image function $I_{\mathrm{DF}}(u,v,\sigma)$.

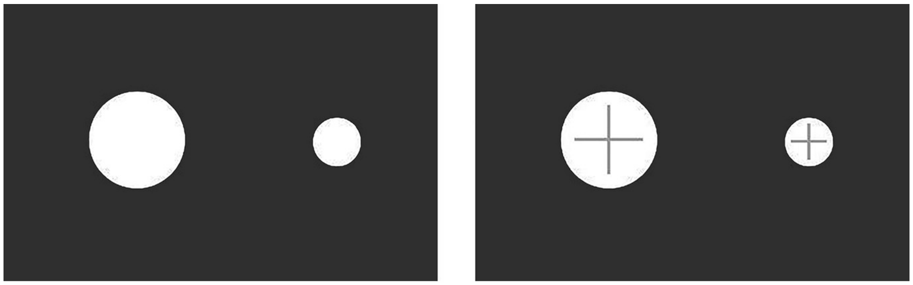
Besides, the last of above properties implies that usage of the scale-invariant Laplacian operator $\sigma^2\Delta$ will neither bias detection output towards smaller blobs nor towards larger blobs. This is why the operator $\sigma^2\Delta$ is called “scale-invariant”.

In summary, the scale-invariant Laplacian operator $\sigma^2\Delta$ possesses the same merit as the Laplacian operator Δ while overcoming the limitation of the Laplacian operator Δ, and tends to outperform the Laplacian operator Δ in practical applications.

It is worth noting that in all above analysis, relevant statements about “positive definite”, “negative definite”, or “minimum” are based on the assumption that the disk blob intensity $I_0 > 0$. If the disk blob intensity $I_0 < 0$, then just reverse “positive definite”, “negative definite”, and “minimum” to “negative definite”, “positive definite”, and “maximum” in relevant statements. This will not influence validity of conclusions about “extremum”.

Detection of Circular Blobs

To demonstrate effects of the scale-invariant Laplacian operator $\sigma^2\Delta$, generate a synthetic image of two circular blobs, as illustrated in Fig. 2.22a.



(a) (b)

Fig. 2.22 Detect blobs via the scale-invariant Laplacian **a** original synthetic image; **b** detected circular blobs

Matlab code for implementing the Laplacian operator $\Delta \equiv \frac{\partial^2}{\partial u^2} + \frac{\partial^2}{\partial v^2}$ is given as follows.

Laplacian2D.m

```
%% Two-dimensional Laplacian
function imgD = Laplacian2D(img)
    H = size(img,1); W = size(img,2);
    imgD = zeros(H+2, W+2); imgD(2:H+1,2:W+1) = double(img);
    Iuu = imgD(2:H+1,3:W+2)-2*imgD(2:H+1,2:W+1)+imgD(2:H+1,1:W);
    Ivv = imgD(3:H+2,2:W+1)-2*imgD(2:H+1,2:W+1)+imgD(1:H,2:W+1);
    imgD = Iuu+Ivv;
end
```

Matlab code for blob detection via the scale-invariant Laplacian operator $\sigma^2\Delta$ is given as follows. The involved sub-function **NonMaximumSuppression3D.m** has already been given previously in Sect. 2.2.4.

LaplacianFindBlob.m

```
%% Blob detection via scale-invariant Laplacian
function blob = LaplacianFindBlob(img, tL, sL)
    [H,W] = size(img); blob = []; if (nargin<3) sL=round(min(H,W
    )/6); end
    ss = 1:sL; sk = length(ss); imgS = zeros(H,W,sk); imgL =
    imgS;
    for k = 1:length(ss)
        imgS(:,:,k) = GaussianFilter2D(img, ss(k));
        imgL(:,:,k) = ss(k)^2*Laplacian2D(imgS(:,:,k));
    end
    imgL = NonMaximumSuppression3D(imgL);
    for k = 2:sk-1
        [vF, uF] = find(abs(imgL(:,:,k)) > tL);
        for i = 1:length(uF)
            blob = [blob, [uF(i); vF(i); ss(k); imgL(vF(i),uF(i)
    ,k)]];
        end
    end
end
```

The function **LaplacianFindBlob.m** is applied to the synthetic image of two circular blobs and the detection results are illustrated in Fig. 2.22b. For each blob, the detected blob center is marked by a cross and the detected σ value that corresponds to the blob radius $\left(\text{i.e. the critical } \sigma \text{ value } \sigma_{\mathrm{c}} = \frac{r_0}{\sqrt{2}}\right)$ is demonstrated by the cross size. The detected blobs are rather intuitively consistent with ground-truths of the blobs.

Besides, quantitative comparison between the detection results and the ground-truths is given in Table 2.1. The pair of u_0 and v_0 denote the image coordinates of the blob center; r_0 denotes the blob radius; σ_{c} denotes the critical σ value that corresponds to the blob radius r_0; $\Delta\hat{I}_{\mathrm{DFm}}$ denotes the extremum value of the scale-invariant Laplacian; I_0 denotes the blob intensity relative to the background.

Quantitative ground-truths of (u_0, v_0, r_0) of the two circular blobs are computed in the following way: Use the *Canny* edge detection method to extract edge points of the two blobs, and fit circles to the extracted edge points to identify circle parameters, which could then be treated as reliable ground-truths of the two blobs.

Table 2.1 Blob detection via scale-invariant Laplacian

	Small blob detected	**Large blob detected**	**Small blob ground-truth**	**Large blob ground-truth**
u_0 **(pixels)**	327.0	131.0	327.3	131.4
v_0 **(pixels)**	135.0	133.0	135.3	133.4
$r_0 = \sqrt{2}\sigma_{\mathrm{c}}$ **(pixels)**	24.0	46.7	23.6	47.2
$\sigma_c = r_0/\sqrt{2}$ **(pixels)**	17.0	33.0	16.7	33.4
$\Delta\hat{I}_{\mathrm{DFm}} = -2I_0/e$	−187.8	−188.4	−187.6	−187.6
$I_0 = -e\Delta\hat{I}_{\mathrm{DFm}}/2$	255.3	256.0	255.0	255.0

The theoretical relationship between the blob radius r_0 and the critical value σ_{c} is

$$r_0 = \sqrt{2}\sigma_{\mathrm{c}} \Longleftrightarrow \sigma_{\mathrm{c}} = \frac{r_0}{\sqrt{2}}.$$

Therefore, once quantitative r_0 ground-truths of the two blobs are obtained in above way, quantitative σ_c ground-truths of the two blobs can be derived accordingly. On the other hand, once σ_c values of the two blobs are obtained by the function **LaplacianFindBlob.m**, corresponding r_0 values of the two blobs can also be derived accordingly.

The theoretical relationship between the extremum value $\Delta\hat{I}_\mathrm{DFm}$ and the blob relative intensity I_0 is

$$\Delta\hat{I}_\mathrm{DFm} = -\frac{2I_0}{e} \Longleftrightarrow I_0 = -\frac{e\Delta\hat{I}_\mathrm{DFm}}{2}.$$

The quantitative I_0
ground-truth of the two blobs is known to be 255 (completely white), and the quantitative $\Delta\hat{I}_\mathrm{DFm}$ ground-truth is derived accordingly (namely $-\frac{2\times255}{e} \approx -187.6$).

Once $\Delta\hat{I}_\mathrm{DFm}$ values of the two blobs are obtained by the function **LaplacianFindBlob.m**, corresponding I_0 values of the two blobs can also be derived accordingly.

As given in Table 2.1, the detected blobs are also quantitatively consistent with ground-truths of the blobs. The small discrepancy between the detection results and the ground-truths is due to numeric approximation, image discretization, and scale-space discretization as well.

2.5.4 Difference-of-Gaussian

As explained in previous sub-sections, given a generic image $I(u, v)$, one can establish its scale space which is represented by the filtered image function $I_\mathrm{F}(u,v,\sigma)$ and then take advantage of the scale-invariant Laplacian operator $\sigma^2\Delta$ to detect salient blobs or blob-like areas in the image.[13] More specifically, compute $\sigma^2\Delta I_\mathrm{F}(u,v,\sigma)$ namely the scale-invariant Laplacian of the filtered image function $I_\mathrm{F}(u,v,\sigma)$. Find extrema of the scale-invariant Laplacian $\sigma^2\Delta I_\mathrm{F}(u,v,\sigma)$, by which one can localize potential blobs and estimate their scales as well.

Instead of directly computing the scale-invariant Laplacian, a convenient and efficient method that exploits the heat equation is proposed in [5]. The parametric Gaussian filter described in (2.32)

$$g(u,v,\sigma) = \frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}}$$

satisfies the heat equation [14]

$$\Delta g(u,v,\sigma) = \frac{1}{2\pi}\mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}}\left(\frac{u^2+v^2}{\sigma^6} - \frac{2}{\sigma^4}\right) = \frac{1}{\sigma}\frac{\partial g(u,v,\sigma)}{\partial\sigma},$$

then

$$\sigma^2\, g(u,v,\sigma) = \sigma\frac{\partial g(u,v,\sigma)}{\partial\sigma} \approx \sigma\frac{g(u,v,\tau\sigma)-g(u,v,\sigma)}{\tau\sigma-\sigma} = \frac{g(u,v,\tau\sigma)-g(u,v,\sigma)}{\tau-1}$$

and

$$\begin{aligned}\sigma^2\Delta I_\mathrm{F}(u,v,\sigma) = {} & \sigma^2\Delta g(u,v,\sigma) * I(u,v) = \sigma\frac{\partial g(u,v,\sigma)}{\partial\sigma} * I(u,v) \\ \approx {} & \frac{g(u,v,\tau\sigma)-g(u,v,\sigma)}{\tau-1} * I(u,v) = \frac{I_\mathrm{F}(u,v,\tau\sigma)-I_\mathrm{F}(u,v,\sigma)}{\tau-1},\end{aligned} \tag{2.49}$$

where τ denotes a geometric series ratio close to 1 in practice—The approximation in (2.49) becomes better and better as τ goes closer and closer to 1, yet a τ value even as large as $\sqrt{2}$ would still work well in practice.

The method described in (2.49) can be interpreted as follows: If the Gaussian filtered images that represent the original image's scale space are generated according to a geometric series of σ parameters respectively, then the Laplacian of the filtered image function $I_{\mathrm{F}}(u, v, \sigma)$ can be fairly approximated by differences of Gaussian filtered images . The method is called the **difference-of-Gaussian** method.

For efficient realization of the difference-of-Gaussian method, one can take advantage of the Gaussian pyramid representation introduced in Sect. 2.5.2 instead of the direct discrete representation. For this purpose, a point to notice is that one needs to generate several extra Gaussian filtered images in each octave besides those just intended for representing and illustrating the octave .

For example, consider the Gaussian pyramid illustrated in Fig. 2.21, where the prior smoothing $\sigma_0 = 1.6$ and the geometric series of σ parameters in each octave of Gaussian filtered images is set as $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0(\text{i. e. } 2\sigma_0)\}$ with $\tau = \sqrt[3]{2} \approx 1.259\,9$. Add two more σ parameters to augment the geometric series to $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0(\text{i. e. } 2\sigma_0), \tau^4\sigma_0, \tau^5\sigma_0\}$, and generate two more Gaussian filtered images according to the two extra σ parameters for each octave . The augmented Gaussian pyramid corresponding to the Gaussian pyramid illustrated in Fig. 2.21 is illustrated in Fig. 2.23.

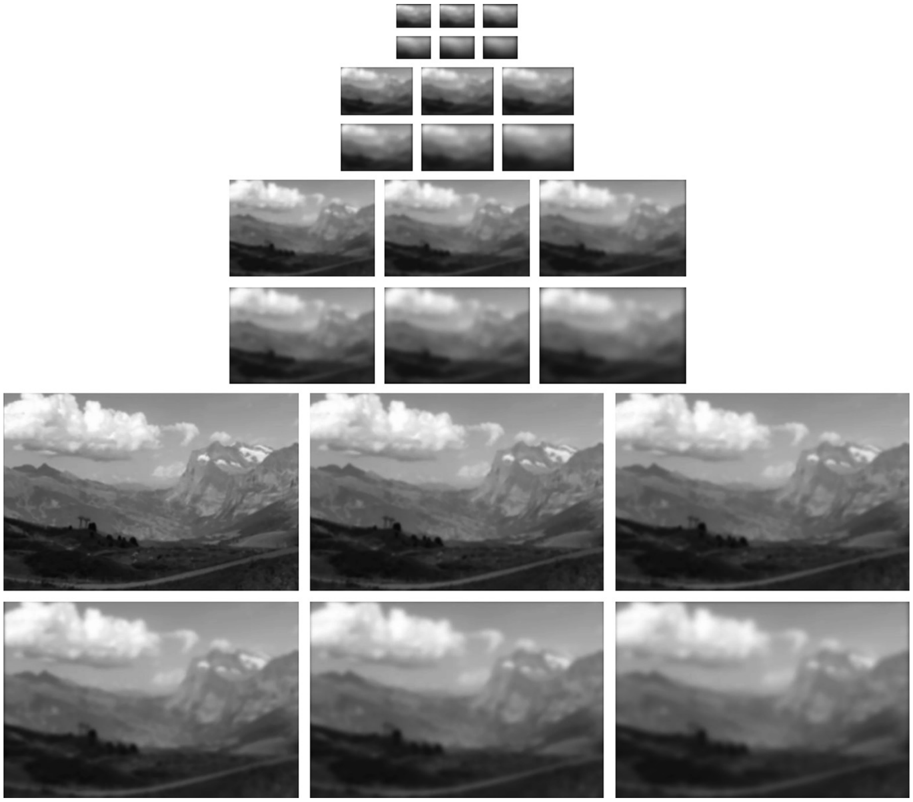


Fig. 2.23 Augmented Gaussian pyramid (prior smoothing $\sigma_0 = 1.6$ and augmented geometric series $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0(\text{i. e. } 2\sigma_0), \tau^4\sigma_0, \tau^5\sigma_0\}$ with $\tau = \sqrt[3]{2}$)

For $\{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \tau^3\sigma_0(\text{i. e. } 2\sigma_0), \tau^4\sigma_0\}$, approximate their associated scale-invariant Laplacian layers [15] via (2.49):

$$\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0) \;\approx\; \frac{I_{\mathrm{F}}(u,v,\tau^{k+1}\sigma_0)-I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\tau-1}, \qquad k\in\{0,1,2,3,4\}.$$

As already clarified in Sect. 2.5.2, for each octave , it is the Gaussian filtered images associated with the three σ parameters $\{\tau\sigma_0,\tau^2\sigma_0,\tau^3\sigma_0\}$ (namely $I_{\mathrm{F}}(u,v,\tau\sigma_0)$, $I_{\mathrm{F}}(u,v,\tau^2\sigma_0)$, and $I_{\mathrm{F}}(u,v,\tau^3\sigma_0)$) that represent the octave of scale space. Consider these Gaussian filtered images ’ scale-invariant Laplacian layers $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau\sigma_0)$, $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^2\sigma_0)$, and $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^3\sigma_0)$.

Find their extrema via non-maximum suppression in scale space. More specifically, given a generic point $(u,v,\tau^k\sigma_0)$ where $k\in\{1,2,3\}$, if $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)$ is neither the maximum nor the minimum within $(u,v,\tau^k\sigma_0)$’s neighbourhood

$$\{(u+\delta u,v+\delta v,\tau^{k+\delta k}\sigma_0)|\delta u,\delta v,\delta k\in\{-1,0,+1\}\},$$

namely if neither

$$\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)=\max\{(u+\delta u,v+\delta v,\tau^{k+\delta k}\sigma_0)|\delta u,\delta v,\delta k\in\{-1,0,+1\}\}$$

nor

$$\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)=\min\{(u+\delta u,v+\delta v,\tau^{k+\delta k}\sigma_0)|\delta u,\delta v,\delta k\in\{-1,0,+1\}\}$$

holds, then suppress $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)$ to zero. The bottom scale-invariant Laplacian layer $\sigma^2\Delta I_{\mathrm{F}}(u,v,\sigma_0)$ and the top scale-invariant Laplacian layer $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^4\sigma_0)$ of the octave are computed just for purpose of non-maximum suppression in scale space.

It is worth noting again that setting $\tau=\sqrt[3]{2}$ is just an example choice. More generally, if one sets $\tau=\sqrt[m]{2}$ where m denotes a generic integer, then the augmented geometric series of σ parameters associated with each octave is

$$S_{\sigma_0,\tau,m}\equiv\{\sigma_0,\tau\sigma_0,\tau^2\sigma_0,\cdots,\tau^{m-1}\sigma_0,\tau^m\sigma_0(\text{i. e. }\ 2\sigma_0),\tau^{m+1}\sigma_0,\tau^{m+2}\sigma_0\}.$$

For $S_{\sigma_0,\tau,m}$, approximate their associated scale-invariant Laplacian layers via (2.49):

$$\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)\;\approx\frac{I_{\mathrm{F}}(u,v,\tau^{k+1}\sigma_0)-I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\tau-1},\quad k\in\{0,1,2,\cdots,m,m+1\}.$$

For each octave , it is the m intermediate Gaussian filtered images $I_{\mathrm{F}}(u,v,\tau\sigma_0)$, $I_{\mathrm{F}}(u,v,\tau^2\sigma_0)$, $\cdots$, $I_{\mathrm{F}}(u,v,\tau^m\sigma_0)$ that represent the octave of scale space. Consider their scale-invariant Laplacian layers $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau\sigma_0)$, $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^2\sigma_0)$, $\cdots$, $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^m\sigma_0)$.

Find extrema of the scale-invariant Laplacian layers via non-maximum suppression in scale space, in the same way as described previously, namely suppressing a generic point $(u,v,\tau^k\sigma_0)$’s scale-invariant Laplacian value $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)$ to zero if $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^k\sigma_0)$ is neither the maximum nor the minimum within $(u,v,\tau^k\sigma_0)$’s neighbourhood

$$\{(u+\delta u,v+\delta v,\tau^{k+\delta k}\sigma_0)|\delta u,\delta v,\delta k\in\{-1,0,+1\}\},$$

where $k\in\{1,2,\cdots,m\}$. The bottom scale-invariant Laplacian layer $\sigma^2\Delta I_{\mathrm{F}}(u,v,\sigma_0)$ and the top scale-invariant Laplacian layer $\sigma^2\Delta I_{\mathrm{F}}(u,v,\tau^{m+1}\sigma_0)$ of the octave are computed just for purpose of non-maximum suppression in scale space.

Scale-invariant Laplacian extrema in all involved octaves of scale space are obtained after non-maximum suppression of $\sigma^2\Delta I_{\mathrm{F}}(u,v,\sigma)$ in scale space. Despite their abstract existence, scale-invariant Laplacian extrema in scale space are important scale-space features that are stable and salient. It is true that scale-invariant Laplacian extrema in scale space are rarely used directly in practical applications, yet they provide a solid base for further extraction of sophisticated features that are applied widely. Details on extraction of such sophisticated features will be presented next in Sect. 2.6.

2.6 Scale Invariant Feature Transform (SIFT)

The sophisticated features just mentioned in Sect. 2.5 are called the **scale invariant feature transform** or, usually in the acronym form, the **SIFT** features [5]. Extraction of SIFT features involve three extra steps, namely feature point localization, local image orientation assignment, and local image descriptor generation, based on extraction of scale-invariant Laplacian extrema in scale space.

2.6.1 Feature Point Localization

Suppose $\tau = \sqrt[m]{2}$ where m denotes a generic integer, then the augmented geometric series of σ parameters associated with each octave is

$$S_{\sigma_0,\tau,m} \equiv \{\sigma_0, \tau\sigma_0, \tau^2\sigma_0, \cdots, \tau^{m-1}\sigma_0, \tau^m\sigma_0(\mathrm{i.e.}\ \ 2\sigma_0), \tau^{m+1}\sigma_0, \tau^{m+2}\sigma_0\}.$$

For each octave , it is the m Gaussian filtered images associated with the geometric series of σ parameters

$$O_{\sigma_0,\tau,m} \equiv \{\tau\sigma_0, \tau^2\sigma_0, \cdots, \tau^{m-1}\sigma_0, \tau^m\sigma_0(\mathrm{i.e.}\ \ 2\sigma_0)\}$$

that represent the octave of scale space.

Suppose the Gaussian pyramid representation is adopted [16] and suppose a generic scale-invariant Laplacian extremum is $\sigma^2\Delta I_{\mathrm{F}}(u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)$ in the n-th octave . If the extremum magnitude is below a predefined threshold t_{SIFT}, namely if

$$|\sigma^2\Delta I_{\mathrm{F}}(u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)| < t_{\mathrm{SIFT}}, \tag{2.50}$$

then discard the extremum.

The extremum position $(u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)$ in scale space is expressed in the image coordinates system of the n-th octave . Note that the scale of the n-th octave's Gaussian filtered images is $1/2$ the scale of the $(n-1)$-th octave's Gaussian filtered images , is $1/4$ the scale of the $(n-2)$-th octave's Gaussian filtered images , ..., is $1/2^{n-1}$ the scale of the first octave's Gaussian filtered images , i.e. $1/2^{n-1}$ the scale of the original image. So the extremum position, if expressed equivalently in the original image coordinates system, is

$$(u_{\mathrm{E}}, v_{\mathrm{E}}, \sigma_{\mathrm{E}}) = (u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)\cdot 2^{n-1} = (2^{n-1}u_{\mathrm{E},n}, 2^{n-1}v_{\mathrm{E},n}, 2^{n-1}\tau^k\sigma_0). \tag{2.51}$$

The equation described in (2.51) gives the basic solution of SIFT feature point localization.

However, this basic solution is rather raw, especially for scale-invariant Laplacian extrema in higher octaves . For example, consider a scale-invariant Laplacian extremum in the fourth octave, then an offset of one pixel for $(u_{\mathrm{E},4}, v_{\mathrm{E},4})$ in the image coordinates system of the fourth octave corresponds to an amplified offset of eight pixels for $(u_{\mathrm{E}}, v_{\mathrm{E}})$ in the original image coordinates system.

To enhance accuracy of SIFT feature point localization, one may take advantage of quadratic interpolation about the scale-invariant Laplacian extremum. For example, consider two-dimensional quadratic interpolation of the (*u*, *v*) coordinates about $(u_{\mathrm{E},n}, v_{\mathrm{E},n})$ in the image coordinates system of the *n*-th octave . Denote the scale-invariant Laplacian layer that contains the extremum shortly as

$$D(\delta u_n, \delta v_n) \equiv \sigma^2 \Delta I_{\mathrm{F}}(u_{\mathrm{E},n} + \delta u_n, v_{\mathrm{E},n} + \delta v_n, \tau^k \sigma_0),$$

where the subscript “*n*” highlights that the (*u*, *v*) coordinates are expressed in the image coordinates system of the *n*-th octave .

Compute the gradient vector and the Hessian matrix of $\sigma^2 \Delta I_{\mathrm{F}}(u_n, v_n, \tau^k \sigma_0)$ at $(u_{\mathrm{E},n}, v_{\mathrm{E},n})$ namely those of $D(\delta u_n, \delta v_n)$ at the origin via numeric approximation:

$$\begin{aligned} D_u &\approx \tfrac{D(1,0)-D(-1,0)}{2}, \quad D_v \approx \tfrac{D(0,1)-D(0,-1)}{2}, \\ D_{uu} &\approx D(1,0)-2D(0,0)+D(-1,0), \quad D_{vv} \approx D(0,1)-2D(0,0)+D(0,-1), \\ D_{uv} &\approx \tfrac{D(1,1)-D(1,-1)-D(-1,1)+D(-1,-1)}{4}. \end{aligned} \tag{2.52}$$

So

$$\begin{aligned} & \sigma^2 \Delta I_{\mathrm{F}}(u_n, v_n, \tau^k \sigma_0) \\ = \; & D(u_n - u_{\mathrm{E},n}, v_n - v_{\mathrm{E},n}), \text{i. e. } D(\delta u_n, \delta v_n) \\ \approx \; & D(0,0) + \nabla D^{\mathrm{T}} \begin{bmatrix} \delta u_n \\ \delta v_n \end{bmatrix} + \tfrac{1}{2} [\delta u_n \quad \delta v_n] \nabla^2 D \begin{bmatrix} \delta u_n \\ \delta v_n \end{bmatrix} \\ = \; & D(0,0) + [D_u \quad D_v] \begin{bmatrix} \delta u_n \\ \delta v_n \end{bmatrix} + \tfrac{1}{2} [\delta u_n \quad \delta v_n] \begin{bmatrix} D_{uu} & D_{uv} \\ D_{uv} & D_{vv} \end{bmatrix} \begin{bmatrix} \delta u_n \\ \delta v_n \end{bmatrix}, \end{aligned} \tag{2.53}$$

which in the quadratic form takes the extremum at

$$\begin{aligned} \begin{bmatrix} \delta \hat{u}_n \\ \delta \hat{v}_n \end{bmatrix} &= -(\nabla^2 D)^{-1} \nabla D = -\begin{bmatrix} D_{uu} & D_{uv} \\ D_{uv} & D_{vv} \end{bmatrix}^{-1} \begin{bmatrix} D_u \\ D_v \end{bmatrix} \\ &= -\tfrac{1}{D_{uu} D_{vv} - D_{uv}^2} \begin{bmatrix} D_{vv} D_u - D_{uv} D_v \\ D_{uu} D_v - D_{uv} D_u \end{bmatrix}. \end{aligned}$$

Then the refined estimate of the SIFT feature point position is

$$\hat{u}_{\mathrm{E},n} = u_{\mathrm{E},n} + \delta \hat{u}_n = u_{\mathrm{E},n} - \tfrac{D_{vv} D_u - D_{uv} D_v}{\det(\nabla^2 D)}, \tag{2.54a}$$

$$\hat{v}_{\mathrm{E},n} = v_{\mathrm{E},n} + \delta \hat{v}_n = v_{\mathrm{E},n} - \tfrac{D_{uu} D_v - D_{uv} D_u}{\det(\nabla^2 D)}. \tag{2.54b}$$

To avoid large deviation of the offset $(\delta \hat{u}_n, \delta \hat{v}_n)$ in (2.54) which occasionally takes place for some scale-invariant Laplacian extrema, one may constrain $\delta \hat{u}_n$ and $\delta \hat{v}_n$ to the range of $[-0.5, 0.5]$. If $\delta \hat{u}_n$ and $\delta \hat{v}_n$ are outside the range, then truncate them into the range.

Besides above example of two-dimensional quadratic interpolation of the (*u*, *v*) coordinates about $(u_{\mathrm{E},n}, v_{\mathrm{E},n})$, one may also follow other kinds of interpolation, such as three-dimensional quadratic interpolation of the (u, v, σ) coordinates about $(u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)$ in the n-th octave of scale space [8]. Such details are omitted here.

As suggested in [5], scale-invariant Laplacian extrema that are caused by edges had better be eliminated, because their positions are sensitive to noises. Since edge-caused extrema tend to have $\nabla^2 D$ with large eigenvalue ratio,[17] then for purpose of eliminating such kinds of extrema, one can discard scale-invariant Laplacian extrema whose $\nabla^2 D$ have the eigenvalue ratio beyond certain threshold, or equivalently whose $\nabla^2 D$ have the following index

$$\frac{\mathrm{tr}(\nabla^2 D)^2}{\det(\nabla^2 D)}$$

beyond certain threshold.[18]

Finally, following (2.51) and (2.54), one can obtain the refined estimate of the SIFT feature point position expressed equivalently in the original image coordinates system:

$$(\hat{u}_{\mathrm{E}}, \hat{v}_{\mathrm{E}}, \sigma_{\mathrm{E}}) = (\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n}, \tau^k\sigma_0) \cdot 2^{n-1} = (2^{n-1}\hat{u}_{\mathrm{E},n}, 2^{n-1}\hat{v}_{\mathrm{E},n}, 2^{n-1}\tau^k\sigma_0). \tag{2.55}$$

2.6.2 Local Image Orientation Assignment

For a generic SIFT feature point, this step consists in assigning a consistent orientation to the SIFT feature point according to its surrounding local image properties and representing its surrounding local image properties with respect to the orientation, which hence enables rotation-invariant representation of its surrounding local image properties.

For a generic scale-invariant Laplacian extremum $\sigma^2\Delta I_{\mathrm{F}}(u_{\mathrm{E},n}, v_{\mathrm{E},n}, \tau^k\sigma_0)$ in the n-th octave , suppose the refined estimate of the SIFT feature point position, namely $(\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n})$, is obtained via (2.54).

Consider a circular local image patch surrounding the center $(\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n})$ with the radius $r_k = 4.5\tau^k\sigma_0$, i.e.

$$C_{\hat{u}_{\mathrm{E},n},\hat{v}_{\mathrm{E},n},k} \equiv \{(u, v) | (u - \hat{u}_{\mathrm{E},n})^2 + (v - \hat{v}_{\mathrm{E},n})^2 \leqslant r_k^2\}.$$

Generate an orientation histogram that are voted by sample pixels in the local image patch $C_{\hat{u}_{\mathrm{E},n},\hat{v}_{\mathrm{E},n},k}$. Suppose the orientation histogram has n_{B} bins that correspond to the n_{B} orientation angles distributed equally within the 2π range. One may set $n_{\mathrm{B}} = 36$ in practice.

Given a generic sample pixel (u, v) in the local image patch $C_{\hat{u}_{\mathrm{E},n},\hat{v}_{\mathrm{E},n},k}$, let

$$I_m^k(u, v) = \sqrt{\left(\frac{\partial I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\partial u}\right)^2 + \left(\frac{\partial I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\partial v}\right)^2}, \tag{2.56a}$$

$$I_a^k(u, v) = \arctan\left(\frac{\partial I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\partial u}, \frac{\partial I_{\mathrm{F}}(u,v,\tau^k\sigma_0)}{\partial v}\right) \tag{2.56b}$$

denote respectively the magnitude and orientation angle of $I_{\mathrm{F}}(u, v, \tau^k\sigma_0)$'s gradient at ($u$, v). The sample pixel (u, v) adds a weighted gradient magnitude

$$I_m^k(u,v)\cdot \mathrm{e}^{-\frac{(u-\hat{u}_{\mathrm{E},n})^2+(v-\hat{v}_{\mathrm{E},n})^2}{2\sigma_k^2}} \quad (\text{where } \sigma_k = 1.5\tau^k\sigma_0) \tag{2.57}$$

to the orientation bin closest to $I_a^k(u,v)$ in the orientation histogram .

The formalism of weighted gradient magnitude described in (2.57) is transformed to

$$I_m^k(\hat{u}_{\mathrm{E},n}+\delta u, \hat{v}_{\mathrm{E},n}+\delta v)\cdot \mathrm{e}^{-\frac{\delta u^2+\delta v^2}{2\sigma_k^2}} \tag{2.58}$$

which better highlights the fact that the sample pixels' contributions to the orientation histogram are weighted by a Gaussian filtering kernel . To enhance computational efficiency, the Gaussian weights

$$\mathrm{e}^{-\frac{\delta u^2+\delta v^2}{2\sigma_k^2}}$$

and their corresponding (u, v) offsets $(\delta u, \delta v)$ for all local image patches associated with SIFT feature points can be pre-computed once and stored in a look-up table for repetitive reference.

The gradients and orientation histogram of the local image patch attached to an example SIFT feature point are illustrated in Figs. 2.24 and 2.25. Figure 2.24a illustrates the original gradients of the circular local image patch, where each square represents a pixel and the line with one end attached to the square center represents the image gradient at the pixel. Figure 2.24b illustrates the Gaussian weighted version of those illustrated in Fig. 2.24a.

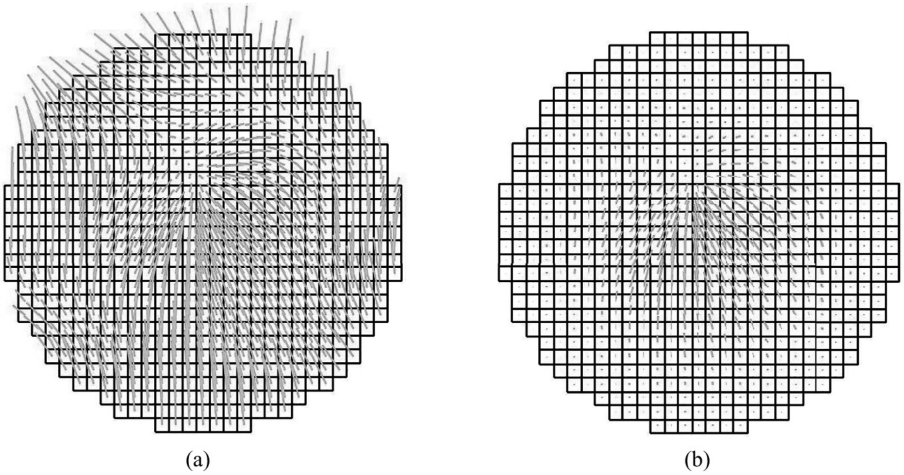

(a)
(b)

Fig. 2.24 Gradients of the local image patch **a** original gradients; **b** Gaussian weighted gradients

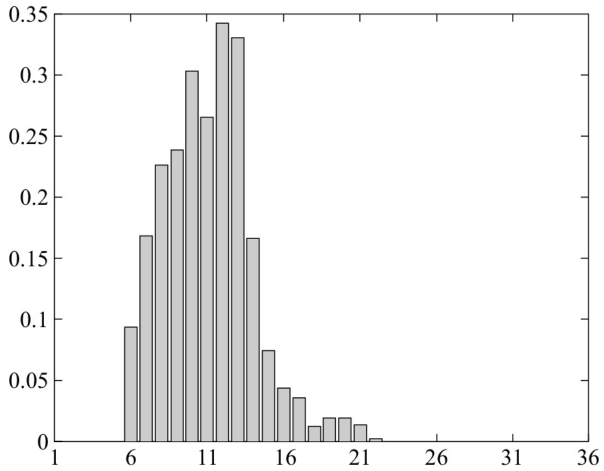


Fig. 2.25 Orientation histogram of Gaussian weighted gradients

Figure 2.25 illustrates the orientation histogram established according to the Gaussian weighted gradients of the circular local image patch.

Once the orientation histogram is established, extract prominent peaks in the orientation histogram which correspond to dominant orientations of image gradients of the local image patch. First extract the highest peak in the orientation histogram and then take all peaks whose heights are beyond 80% that of the highest peak. Among the extracted candidate peaks, discard those that are lower than at least one of their two immediate neighbour bins, for sake of sifting out all candidate peaks whose local maximality is violated. Orientations associated with remained peak bins are all accepted as valid orientations of the SIFT feature point.

When a SIFT feature point has a unique valid orientation, then incorporate the orientation as extra property into the SIFT feature point. When a SIFT feature point has multiple valid orientations, then generate the same number of copies of the SIFT feature point namely the same number of new SIFT feature points that are identical to the original SIFT feature point, and incorporate the orientations as extra property into the new SIFT feature points respectively.

For example, see the orientation histogram illustrated in Fig. 2.25. The highest peak corresponds to the 12th orientation bin. The bars in the 10th and 13th orientation bins are beyond 80% the height of the highest peak and hence are taken as candidate peaks. The candidate peak in the 13th orientation bin is lower than its left neighbour in the 12th orientation bin, which violates its local maximality. Then it is discarded. Finally, orientations associated with the 10th and 12th orientation bins are accepted as valid orientations of the SIFT feature point, and are incorporated as extra property into the SIFT feature point respectively to form two new SIFT feature points.

Besides, for better localization of each valid orientation, one may take advantage of quadratic interpolation of the orientation bin itself together with its two neighbour bins. Suppose the bin index of a valid orientation is b_{M}, i.e. the orientation is $2(b_{\mathrm{M}}-1)\pi/n_{\mathrm{B}}$, and the histogram values associated with the bins $b_{\mathrm{M}}-1$, b_{M}, and $b_{\mathrm{M}}+1$ are h_{L}, h_{M}, and h_{R} respectively.

Similar to one-dimensional quadratic interpolation (2.13) presented in Sect. 2.1.3, one has the interpolated histogram value function $h(b)$ in terms of an imagined real-value bin index variable b:

$$h(b) = \quad (b-b_{\mathrm{M}})^2\,\tfrac{h_{\mathrm{R}}+h_{\mathrm{L}}-2h_{\mathrm{M}}}{2} + (b-b_{\mathrm{M}})\,\tfrac{h_{\mathrm{R}}-h_{\mathrm{L}}}{2} + h_{\mathrm{M}},$$

from which one can compute the optimal bin index b_{opt} as

$$b_{\text{opt}} = b_{\text{M}} + \frac{h_{\text{R}} - h_{\text{L}}}{2(h_{\text{M}} - h_{\text{R}} - h_{\text{L}})}, \tag{2.59}$$

namely the refined estimate of the orientation is $2(b_{\text{opt}} - 1)\pi/n_{\text{B}}$.

Still take the original image illustrated in Fig. 2.20 as example, extract SIFT feature points from the original image. The positions, scales, and orientations of only four hundred SIFT feature points among all those extracted are illustrated in Fig. 2.26—If all the extracted SIFT points were visualized in Fig. 2.26, they would crowd the image severely and cause a dizzying visualization effect.

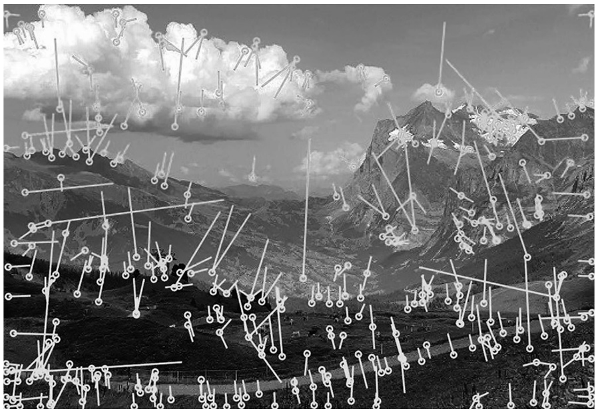


Fig. 2.26 Positions, scales and orientations of SIFT feature points

The position, scale, and orientation of each SIFT feature point is illustrated by a line segment together with a small circle attached at one of its ends: The small circle illustrates the position of the SIFT feature point. The line segment length illustrates the scale of the SIFT feature point.[19] The direction pointed from the small circle to the other end of the line segment illustrates the orientation of the SIFT feature point.

Scale (shrink or expand) and rotate the original image illustrated in Fig. 2.20 to form a scaled-rotated version of the original image, as illustrated in Fig. 2.27b—Here, the original image is actually shrunk for demonstration, yet similar kind of demonstration would also be made if the original image was expanded—Extract SIFT feature points from the original image, as already illustrated in Fig. 2.26, and also extract SIFT feature points from the scaled-rotated version of the original image. The SIFT feature points of the two images are put into correspondence, as partially illustrated in Fig. 2.27.

Figure 2.27a illustrates SIFT feature points extracted from the original image, whereas Fig. 2.27b illustrates SIFT feature points extracted from the scaled-rotated version of the original image. The illustrated SIFT feature points, which are put into correspondence between the two sub-figures, are only part of those extracted from both images respectively.

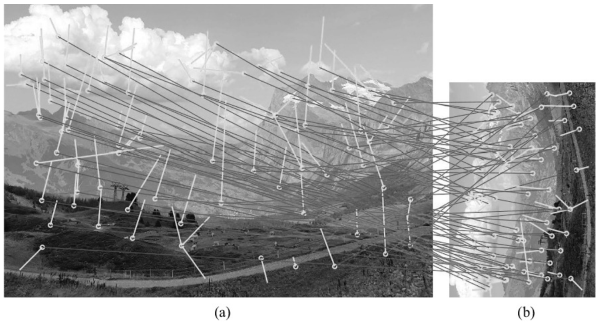


Fig. 2.27 Scale and rotation invariance **a** SIFT feature points extracted from the original image; **b** SIFT feature points extracted from the scaled-rotated image

Although corresponding SIFT feature points have different positions, scales, and orientations in the original image and the scaled-rotated image, one can intuitively find that corresponding SIFT feature points are actually located in the same positions in the real-world scenario, and have the same scales and orientations with respect to the real-world scenario in both images. In other words, extraction of SIFT feature points tends to depend only on the real-world scenario itself, and is not susceptible to scale and rotation differences that exist in images of the same real-world scenario. SIFT feature points' such merit of **scale and rotation invariance** makes them beneficial to a wide range of practical applications that involve establishment of correspondences among different images of the same environment.

2.6.3 Local Image Descriptor Generation

Readers may notice a crucial problem skipped in the previous sub-section: given SIFT feature points extracted respectively from the two images illustrated in Fig. 2.27, how are they put into correspondence?

The kinds of SIFT feature points obtained by so far, which contain only the position, scale, and orientation properties, indeed do not allow easy establishment of SIFT feature point correspondence. It is right the last important step of SIFT feature extraction namely **local image descriptor** generation that largely facilitates establishment of SIFT feature point correspondence .[20]

For each SIFT feature point, the spirit of local image descriptor generation consists in analysing image gradient statistics around the SIFT feature point to form a stable structure that tends to uniquely characterize the SIFT feature point.

Given a generic SIFT feature point associated with the Gaussian filtered image $I_{\mathrm{F}}(u, v, \tau^k \sigma_0)$ in the n-th octave , suppose its position $(\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n})$ is obtained via (2.54) and its orientation angle $\hat{a}_{\mathrm{E},n}$ is obtained via the orientation histogram method introduced in the previous sub-section.

Consider a $4S$-by-$4S$ sample pixel array (i.e. a local image sample patch) surrounding the position $(\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n})$. Pixels in the $4S$-by-$4S$ sample pixel array are weighted by a Gaussian filtering kernel with center $(\hat{u}_{\mathrm{E},n}, \hat{v}_{\mathrm{E},n})$ and with σ parameter σ_S set to $2S$, i.e. a half of the width of the local image sample patch. Besides, to achieve rotation-invariance, the $4S$-by-$4S$ sample pixel array is always aligned in the SIFT feature point's orientation, as if it is obtained by rotating the standard one, i.e. the horizontally-aligned $4S$-by-$4S$ sample pixel array by the angle $\hat{a}_{\mathrm{E},n}$.

Divide the $4S$-by-$4S$ sample pixel array into S-by-S sample blocks of the same size, then each sample block consists of 4-by-4 sample pixels. For each sample block, establish an orientation histogram of eight bins that correspond to the eight orientation angles distributed equally within the 2π range. Similar to how the orientation histogram is established for local image orientation assignment, here, each sample pixel contributes a Gaussian weighted gradient magnitude to the orientation histogram of its associated sample block. Following recommendation of [5], *trilinear interpolation* is adopted for establishment of the sample blocks' orientation histograms.

More specifically, given a generic sample pixel $(\delta u, \delta v)$ in the $4S$-by-$4S$ sample pixel array aligned in the SIFT feature point's orientation, compute its counterpart (u, v) in the image coordinates system of the Gaussian filtered image $I_{\mathrm{F}}(u, v, \tau^k \sigma_0)$: rotate it by the orientation angle $\hat{a}_{\mathrm{E},n}$:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \tau^{k-1} \begin{bmatrix} \cos \hat{a}_{\mathrm{E},n} & -\sin \hat{a}_{\mathrm{E},n} \\ \sin \hat{a}_{\mathrm{E},n} & \cos \hat{a}_{\mathrm{E},n} \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} + \begin{bmatrix} \hat{u}_{\mathrm{E},n} \\ \hat{v}_{\mathrm{E},n} \end{bmatrix}, \tag{2.60}$$

where

$$\delta u, \delta v \in \left\{-2S+\tfrac{1}{2}, -2S+\tfrac{3}{2}, \cdots, -\tfrac{3}{2}, -\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{3}{2}, \cdots, 2S-\tfrac{3}{2}, 2S-\tfrac{1}{2}\right\}.$$

For the SIFT feature point, besides rotation for sake of alignment consistent with its orientation, the sample pixel array is also scaled adaptively according to its relative scale in its associated octave, as reflected by the scaling factor τ^{k-1} in (2.60). Such scaling is reasonable in the sense that the SIFT feature point and the sample pixel array for establishing its descriptor would have consistent scales in scale space. In summary, (2.60) conveys that the sample pixel array for establishing the local image descriptor of the SIFT feature point is a rotated and scaled version of the standard $4S$-by-$4S$ sample pixel array.

Note that u and v are usually fractions. The four integer pixels that enclose (u, v) immediately, namely

$$S_{(u,v)} \equiv \{(\lfloor u\rfloor, \lfloor v\rfloor), (\lfloor u\rfloor+1, \lfloor v\rfloor), (\lfloor u\rfloor+1, \lfloor v\rfloor+1), (\lfloor u\rfloor, \lfloor v\rfloor+1)\},$$

all contribute to the orientation histogram of $(\delta u, \delta v)$'s associated sample block.

For each pixel $(\bar{u}, \bar{v})$ in $S_{(u,v)}$, shortly denote $\Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v}) = I_{\mathrm{a}}^{k}(\bar{u}, \bar{v}) - \hat{a}_{\mathrm{E},n}$ and

$$\left\lfloor \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rfloor = \frac{2\pi}{n_{\mathrm{BD}}}\left\lfloor \frac{\Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})}{2\pi/n_{\mathrm{BD}}}\right\rfloor, \quad \left\lceil \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rceil = \frac{2\pi}{n_{\mathrm{BD}}}\left(\left\lfloor \frac{\Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})}{2\pi/n_{\mathrm{BD}}}\right\rfloor + 1\right),$$

where n_{BD} denotes the number of bins in the orientation histogram of each sample block, and is conventionally set to eight, as already mentioned above.

The pixel $(\bar{u}, \bar{v})$ adds a weighted gradient magnitude

$$I_{\mathrm{m}}^{k}(\bar{u}, \bar{v}) \cdot \mathrm{e}^{-\frac{\delta u^2+\delta v^2}{2\sigma_S^2}} \cdot (1-|\bar{u}-u|)(1-|\bar{v}-v|)\left(\frac{\left\lceil \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rceil - \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})}{2\pi/n_{\mathrm{BD}}}\right) \tag{2.61}$$

and another weighted gradient magnitude

$$I_{\mathrm{m}}^{k}(\bar{u}, \bar{v}) \cdot \mathrm{e}^{-\frac{\delta u^2+\delta v^2}{2\sigma_S^2}} \cdot (1-|\bar{u}-u|)(1-|\bar{v}-v|)\left(\frac{\Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v}) - \left\lfloor \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rfloor}{2\pi/n_{\mathrm{BD}}}\right) \tag{2.62}$$

to the two orientation bins associated with $\left\lfloor \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rfloor$ and $\left\lceil \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rceil$ respectively.

The factor

$$\mathrm{e}^{-\frac{\delta u^2+\delta v^2}{2\sigma_S^2}}$$

in (2.61) and (2.62) is the Gaussian weight associated with the sample pixel $(\delta u, \delta v)$ in the $4S$-by-$4S$ sample pixel array—Gaussian weights associated with sample pixels in the $4S$-by-$4S$ sample pixel array can be pre-computed once and stored in a look-up table for repetitive reference—The factors

$$(1-|\bar{u}-u|)(1-|\bar{v}-v|)\left(\frac{\left\lceil \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})\right\rceil - \Delta I_{\mathrm{a}}^{k}(\bar{u}, \bar{v})}{2\pi/n_{\mathrm{BD}}}\right)$$

and

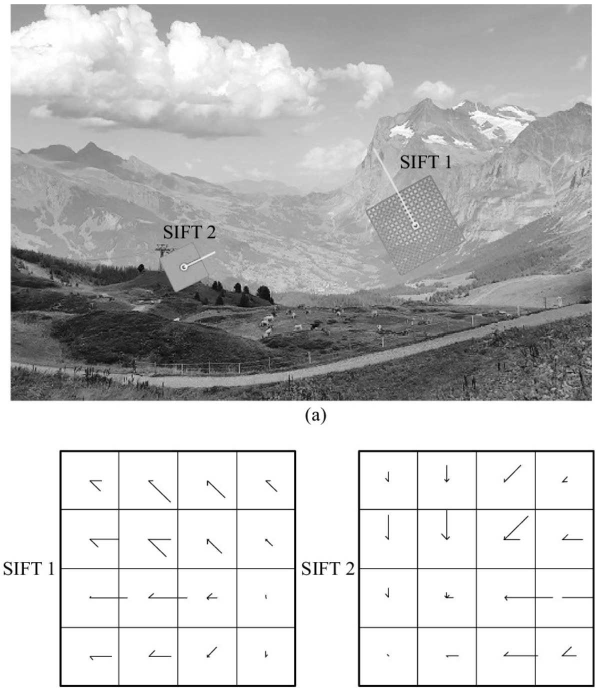
$$(1-|\bar{u}-u|)(1-|\bar{v}-v|)\left(\frac{\Delta I_{\mathrm{a}}^{k}(\bar{u},\bar{v})-\lfloor\Delta I_{\mathrm{a}}^{k}(\bar{u},\bar{v})\rfloor}{2\pi/n_{\mathrm{BD}}}\right)$$

in (2.61) and (2.62) are trilinear interpolation weights associated with the pixel $(\bar{u},\bar{v})$.

An orientation histogram of eight bins is established for each sample block, so a total number of S^2 orientation histograms are established for the local image sample patch. Treat each orientation histogram as a small vector of 8 dimensions and concatenate all the S^2 orientation histograms into a unified large vector of $8S^2$ dimensions. The parameter S is conventionally set to four, so after orientation histogram concatenation, a unified large vector of 128 dimensions is obtained for the SIFT feature point.

Normalize the unified vector of $8S^2$ dimensions, truncate vector elements by the upper limit 0.2 (proposed by [5]), and then normalize the truncated vector again. The unified vector finally obtained is incorporated as extra property into the SIFT feature point and is called the **local image descriptor** or shortly **descriptor** of the SIFT feature point.

The 16-by-16 sample pixel arrays of two example SIFT feature points and orientation histograms of their 4-by-4 sample blocks are illustrated in Fig. 2.28. Each SIFT feature point's 16-by-16 sample pixel array is always aligned in its orientation. It is worth noting that for different SIFT feature points, their 16-by-16 sample pixel arrays are taken from their corresponding Gaussian filtered images in relevant octaves respectively, and can be of different scales from the original image's perspective.

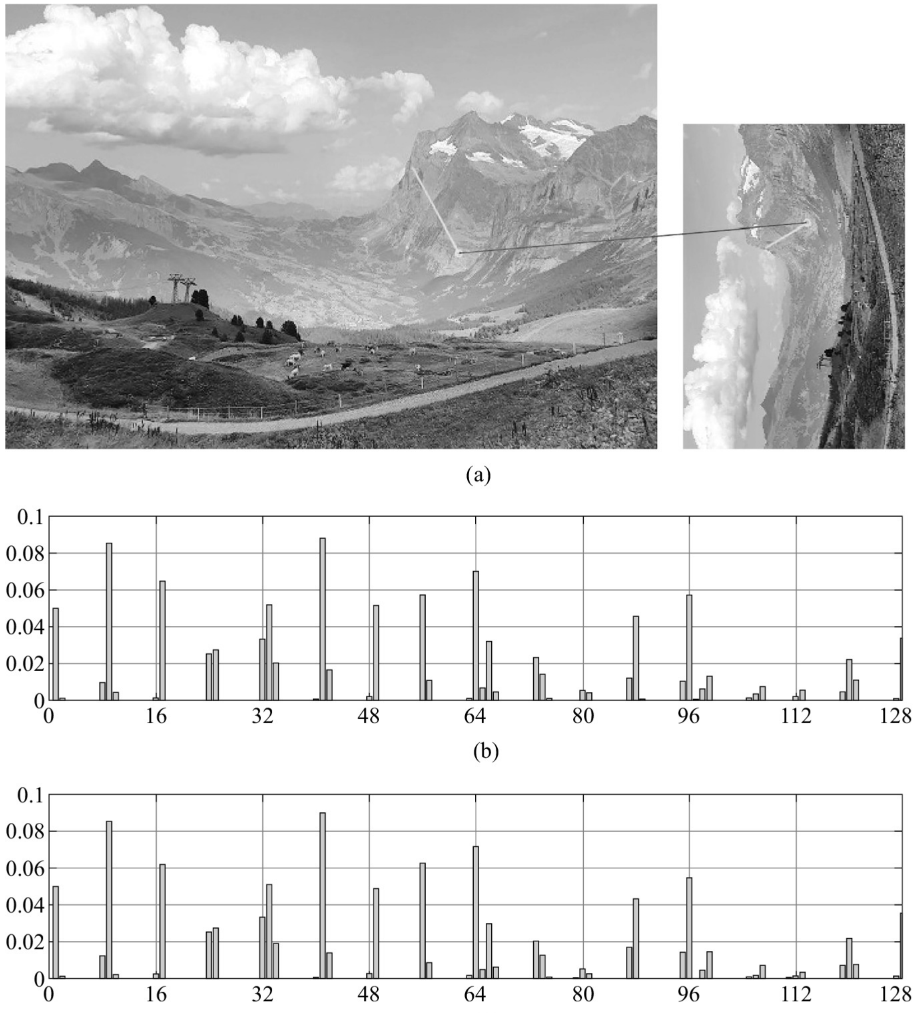


(a)

(b)

Fig. 2.28 Local image descriptors **a** 16-by-16 sample pixel array; **b** normalized orientation histograms of sample blocks

A pair of corresponding SIFT feature points among those illustrated in Fig. 2.27 and the two SIFT feature points' respective local image descriptors are illustrated in Fig. 2.29. Figure 2.29b illustrates the 128-dimensional local image descriptor of the SIFT feature point in the larger image, whereas Fig. 2.29c illustrates the 128-dimensional local image descriptor of the SIFT feature point in the smaller image.



(a)

(b)

(c)

Fig. 2.29 SIFT descriptor consistency **a** a pair of corresponding SIFT feature points; **b** descriptor of the SIFT feature point in the larger image; **c** descriptor of the SIFT feature point in the smaller image

Despite the scale and rotation difference between the two images, the pair of corresponding SIFT feature points, which are located in the same position and point to the same direction in the real-world scenario, have almost the same local image descriptor. In other words, SIFT feature points' local image descriptors possess the merit of **scale and rotation invariance** from the perspective of the real-world scenario. This important merit reflects that SIFT feature points' local image descriptors encode well certain inherent properties of representative parts in the real-world scenario, though SIFT descriptors are indeed abstract and how they encode inherent properties of real-world scenario parts are not so intuitive to humans' biological vision systems. Not only in the example case demonstrated in Fig. 2.29, corresponding SIFT feature points tend to have almost the same local image descriptor in practical applications.

Thanks to above merit of SIFT feature points, a simple yet effective method can be adopted to match corresponding SIFT feature points among different images of the same real-world scenario[21]: Define a distance metric for the 128-dimensional abstract feature space of SIFT descriptors (say L_1-norm distance or Euclidean distance) and treat it as the distance metric for SIFT feature points. Then for a generic SIFT feature point in one image, just match it with the "closest" SIFT feature point among those in another image. Besides, such matching can be performed mutually between two images, namely for SIFT feature points in the first image, match their "closest" counterparts respectively in the second image, and then for SIFT feature points in the second image, also match their “closest” counterparts respectively in the first image. It is when matched SIFT feature points are mutually “closest” to each other that they are kept and regarded as candidate corresponding SIFT feature points. Furthermore, a distance upper limit can be set, if a pair of mutually “closest” SIFT feature points have a distance larger than the distance upper limit, then they will still not be regarded as corresponding SIFT feature points. This last criterion aims at avoiding wrong matching of SIFT feature points that actually do not have their corresponding counterparts in other images.

An application example of the matching method described above has already been illustrated in Fig. 2.27, where the SIFT feature point correspondence between the two sub-figures is established right using the matching method.

Implementation

Finally, the author has finished presentation of knowledge on the SIFT feature, one of the most important and representative kinds of images features in the kingdom of computer vision. In summary, SIFT feature points, which contain properties of position, scale, orientation, and local image descriptor, form a powerful tool for characterizing inherent geometric entities of the real-world scenario, though such geometric entities are implicit and abstract.

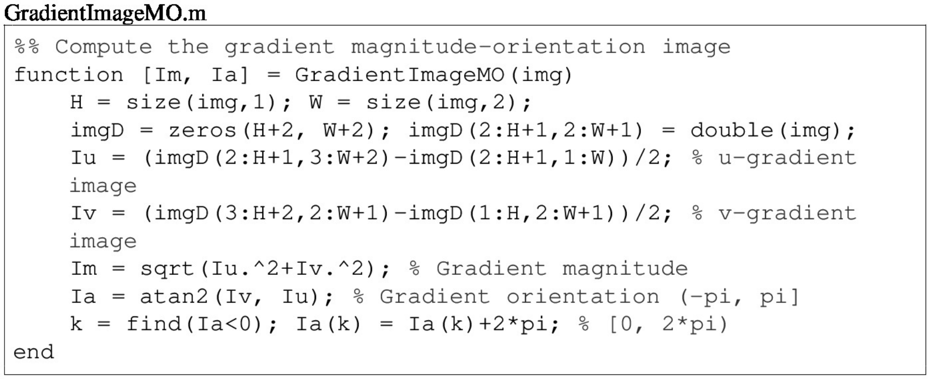
Matlab code for SIFT feature extraction is given as follow.

SIFT.m

```
%% Scale Invariant Feature Transform (SIFT)
% fL: row 1: u-coordinates of SIFT points in corresponding
    Gaussian images
%      row 2: v-coordinates of SIFT points in corresponding
    Gaussian images
%      row 3: Levels of SIFT points' corresponding Gaussian
    images
%      row 4: SIFT points' sigma levels in corresponding Gaussian
     images
%      row 5: u-coordinates of SIFT points in the original image
%      row 6: v-coordinates of SIFT points in the original image
%      row 7: SIFT points' sigmas (reflecting SIFT points' scales
    )
%      row 8: SIFT points' Gaussian image difference values
%      row 9: SIFT points' orientation angles in [0, 2*pi)
% fD: SIFT points' corresponding (vector) descriptors
% oG: Octaves of Gaussian images
% oDG:Octaves of differences of Gaussian images
% img:In Matlab, an image's origin is at the top-left. The row (
    vertical)
%      dimension goes downward, opposite to the upward vertical
    direction in
%      the u-v image coordinates system conforming to right-hand
    conventions
%      As SIFT involves rotation in the conventional u-v ICS, a
    transform is
%      needed in Matlab to make the image origin at the bottom-
    left and the
%      row (vertical) dimension go upward as in the conventional
    u-v ICS, by
%           img = img(end:-1:1,:); % For conventional u-v ICS
    consistency
%      then invoke the SIFT function  [fL, fD, oG, oDG] = SIFT(
    img, ...)
function [fL, fD, oG, oDG] = SIFT(img, tSIFT, s, sigma, rT, mHW)
    %% SIFT parameters
    % tSIFT: Threshold for local extrema detection
    % s: Number of scales per octave
    % sigma: Prior smoothing for each octave
    % rT: Local Hessian matrix root ratio
    % mHW: Minimum allowable image size for octaves of Gaussian
    images
    if (nargin<2) tSIFT = 0.5; s = 3; sigma = 1.6; rT = 10; mHW
    = 64;
    elseif (nargin<3) s = 3; sigma = 1.6; rT = 10; mHW = 64; end
    sk = 2^(1/s); sigmaG = sigma*sk.^[0:(s+2)]; % Sigmas in each
     octave
    tdT = (rT+1)^2/rT; % Threshold for edge response elimination
    %% Reshape image sizes for convenient pyramid processing
    [Horg, Worg] = size(img); sN = max(floor(log2(max(Horg,Worg)
    /mHW)),2);
    sN2 = 2^sN; H0 = floor(Horg/sN2); W0 = floor(Worg/sN2);
    H = H0*sN2; W = W0*sN2; imgA = img(1:H,1:W);
    %% Generate Gaussian pyramid, i.e. octaves of scale space
    oG = []; % Octaves of Gaussian images
    while (H>=H0)
        o.Img = zeros(H,W,s+3); o.Im = o.Img; o.Ia = o.Img;
        for k = 1:s+3
            % Gaussian filtering with incrementally variant
    sigma
            o.Img(:,:,k) = GaussianFilter2D(imgA, sigmaG(k));
            % Gradient magnitude-orientation of each Gaussian
    image
            [o.Im(:,:,k), o.Ia(:,:,k)] = GradientImageMO(o.Img
    (:,:,k));
        end
        oG = [oG, o];
        % Resampling of the Gaussian image with twice sigma
        imgA = o.Img(2:2:end,2:2:end,s+1); H = H/2; W = W/2;
    end
    oDG = []; % Octaves of differences of Gaussian images
    for ok = 1:length(oG)
        [H, W] = size(oG(ok).Img(:,:,1)); o.Img = zeros(H,W,s+2)
    ;
        Hm = 2:H-1; Wm = 2:W-1;
        o.Img(Hm,Wm,:) = oG(ok).Img(Hm,Wm,2:s+3)-oG(ok).Img(Hm,
    Wm,1:s+2);
        oDG = [oDG, o];
    end
    %% Scale-space extrema detection via non-maximum suppression
    oDGorg = oDG;
    for ok = 1:length(oDG)
        oDG(ok).Img = NonMaximumSuppression3D(oDG(ok).Img);
    end
    %% Feature point localization
    fL = [];
    for ok = 1:length(oDG)
        for k = 2:s+1
            % Thresholding
            [vF, uF] = find(abs(oDG(ok).Img(:,:,k)) > tSIFT);
            D = oDGorg(ok).Img(:,:,k); dgF = zeros(size(uF));
            i = 1; i2 = length(uF);
            while (i<=i2)
                u = uF(i); v = vF(i);
                Du = (D(v,u+1)-D(v,u-1))/2; Dv = (D(v+1,u)-D(v
    -1,u))/2;
                Duu = D(v,u+1)-2*D(v,u)+D(v,u-1);
                Dvv = D(v+1,u)-2*D(v,u)+D(v-1,u);
                Duv = (D(v+1,u+1)-D(v-1,u+1)-D(v+1,u-1)+D(v-1,u
    -1))/4;
                trD = Duu+Dvv; detD = Duu*Dvv-Duv^2;
                if ((trD^2/detD)>tdT) % Eliminate edge responses
                    utmp = uF(i2); vtmp = vF(i2);
                    uF(i2) = uF(i); vF(i2) = vF(i);
                    uF(i) = utmp; vF(i) = vtmp; i2 = i2-1;
                else
```

```
                    dgF(i) = D(vF(i),uF(i));
                    % Location interpolation
                    uOff = min(max(-(Dvv*Du-Duv*Dv)/detD, -0.5),
     0.5);
                    vOff = min(max(-(Duu*Dv-Duv*Du)/detD, -0.5),
     0.5);
                    uF(i) = u+uOff; vF(i) = v+vOff; i = i+1;
                end
            end
            uF = uF(1:i2); vF = vF(1:i2); dgF = dgF(1:i2);
            a1 = ones(size(uF')); ok2 = 2^(ok-1); sigma2 =
    sigmaG(k)*ok2;
            p = [uF'; vF'; ok*a1; k*a1; uF'*ok2; vF'*ok2; sigma2
    *a1; dgF'];
            fL = [fL, p];
        end
    end
    oDG = oDGorg;
    %% Local image orientation assignment
    fG = []; % Precomputation of Gaussian weights
    for k = 1:s+3
        stdF = 1.5*sigmaG(k); stdF2 = (3*stdF)^2; n = ceil(3*
    stdF);
        r = ((1:2*n+1)'-(n+1))*ones(1,2*n+1); c = r';
        rc2 = r.^2+c.^2; w = exp(-rc2/(2*stdF^2)); i = find(rc2<
    stdF2);
        f.rOff = r(i); f.cOff = c(i); f.w = w(i); f.w = f.w/sum(
    f.w);
        fG = [fG, f];
    end
    nB = 36; nForg = size(fL,2); fL = [fL; zeros(1,nForg)];
    for i = 1:nForg
        uF = round(fL(1,i)); vF = round(fL(2,i)); ok = fL(3,i);
    k = fL(4,i);
        Im = oG(ok).Im(:,:,k); Ia = oG(ok).Ia(:,:,k); [H,W] =
    size(Im);
        rOff = fG(k).rOff; cOff = fG(k).cOff; w = fG(k).w;
        hB = zeros(1,nB); % Weighted orientation histogram
        for i2 = 1:length(rOff)
            r = vF+rOff(i2); c = uF+cOff(i2);
            if (r>=1 && r<=H && c>=1 && c<=W)
                iB = floor(nB*Ia(r,c)/(2*pi))+1;
                hB(iB) = hB(iB)+w(i2)*Im(r,c);
            end
        end
        % Orientation histogram peaks (potentially multiple)
        angMax = max(hB); iAng = find(hB>0.8*angMax); iM = 0;
        for i2 = iAng
            if (1==i2) hM = hB(1); hL = hB(nB); hR = hB(2);
            elseif (nB==i2) hM = hB(nB); hL = hB(nB-1); hR = hB
    (1);
            else hM = hB(i2); hL = hB(i2-1); hR = hB(i2+1); end
            % If the bin's local maximality is violated, just
    discard it
            if (hM<hL || hM<hR) continue; end
            % Orientation interpolation
            aM = (2*i2-1)*pi/nB + (pi/nB)*((hR-hL)/(2*hM-hL-hR))
    ;
            iM = iM+1;
            if (1==iM) fL(end,i) = aM;
            else fL = [fL, [fL(1:end-1,i); aM]]; end
        end
    end
    %% Local image descriptor generation
    % Precomputation of Gaussian weights
    nS = 4; nD = 16; rS = (-(nD-1)/2:(nD-1)/2)'*ones(1,nD); cS =
     rS';
    rD = zeros(1,nD^2); cD = rD;
    for c = 1:(nD/nS)
        for r = 1:(nD/nS)
            i = r+(c-1)*nD/nS; iD = (i-1)*nS^2+1:i*nS^2;
            ir = (r-1)*nS+1:r*nS; ic = (c-1)*nS+1:c*nS;
            rD(iD) = reshape(rS(ir,ic),1,[]);
            cD(iD) = reshape(cS(ir,ic),1,[]);
        end
    end
    % Gaussian weighting sigma is half the descriptor window
    size
    rcD2 = rD.^2+cD.^2; wD = exp(-rcD2/(2*(nD/2)^2)); wD = wD/
    sum(wD);
    nBD = 8; fD = zeros(nBD*(nD/nS)^2,size(fL,2));
    for i = 1:size(fL,2)
        uvF = fL(1:2,i); ok = fL(3,i); k = fL(4,i); a = fL(9,i);
        Im = oG(ok).Im(:,:,k); Ia = oG(ok).Ia(:,:,k); [H,W] =
    size(Im);
        rotM = [cos(a), -sin(a); sin(a), cos(a)];
        % Scale sample array adaptively according to SIFT sigma
    in octave
        uvF = (sigmaG(k)/sigmaG(2))*rotM*[cD; rD]+uvF*ones(size(
    cD));
        hBD = zeros(nBD, (nD/nS)^2); % Weighted orientation
    histogram
        for i2 = 1:length(cD)
            c = uvF(1,i2); r = uvF(2,i2); iD = ceil(i2/nS^2);
            if (r<=1 || r>=H || c<=1 || c>=W) continue; end
            % Trilinear interpolation: image coordinates and
    orientation bin
            c1 = floor(c); c2 = c1+1; r1 = floor(r); r2 = r1+1;
            rc = [c1, r1, (c2-c)*(r2-r); c2, r1, (c-c1)*(r2-r);
    ...
                c1, r2, (c2-c)*(r-r1); c2, r2, (c-c1)*(r-r1)];
            for i3 = 1:4
                c = rc(i3,1); r = rc(i3,2); w = rc(i3,3)*wD(i2);
                aI = Ia(r,c)-a; if (aI<0) aI = aI+2*pi; end
                aI = nBD*aI/(2*pi); aI1 = floor(aI); aI2 = aI1
    +1;
                iB1 = aI1+1; iB2 = rem(aI2,nBD)+1;
                hBD(iB1,iD) = hBD(iB1,iD)+(aI2-aI)*w*Im(r,c);
                hBD(iB2,iD) = hBD(iB2,iD)+(aI-aI1)*w*Im(r,c);
            end
        end
        fD(:,i) = reshape(hBD,[],1);
        fD(:,i) = fD(:,i)/sum(fD(:,i)); % Normalization
        i2 = find(fD(:,i)>0.2); % Truncated by 0.2
        if (~ isempty(i2))
            fD(i2,i) = 0.2;
            fD(:,i) = fD(:,i)/sum(fD(:,i)); % Re-normalization
        end
    end
end
```

The involved sub-functions **GaussianFilter2D** and **NonMaximumSuppression3D** have already been given in previous sub-sections. The involved sub-function **GradientImageMO** is given as follows.



GradientImageMO.m

```
%% Compute the gradient magnitude-orientation image
function [Im, Ia] = GradientImageMO(img)
    H = size(img,1); W = size(img,2);
    imgD = zeros(H+2, W+2); imgD(2:H+1,2:W+1) = double(img);
    Iu = (imgD(2:H+1,3:W+2)-imgD(2:H+1,1:W))/2; % u-gradient
    image
    Iv = (imgD(3:H+2,2:W+1)-imgD(1:H,2:W+1))/2; % v-gradient
    image
    Im = sqrt(Iu.^2+Iv.^2); % Gradient magnitude
    Ia = atan2(Iv, Iu); % Gradient orientation (-pi, pi]
    k = find(Ia<0); Ia(k) = Ia(k)+2*pi; % [0, 2*pi)
end
```

For implementation in Matlab code, it is worth noting that an image's origin is by default at the top-left of the image array. The row (vertical) dimension goes downward, opposite to the upward vertical direction in the (u, v) image coordinates system conforming to the right-hand conventions. As SIFT related computation involves rotation in the conventional (u, v) image coordinates system, a transform is needed in Matlab to make the image origin be at the bottom-left and the row (vertical) dimension go upward as in the conventional (u, v) image coordinates system.

Discussion

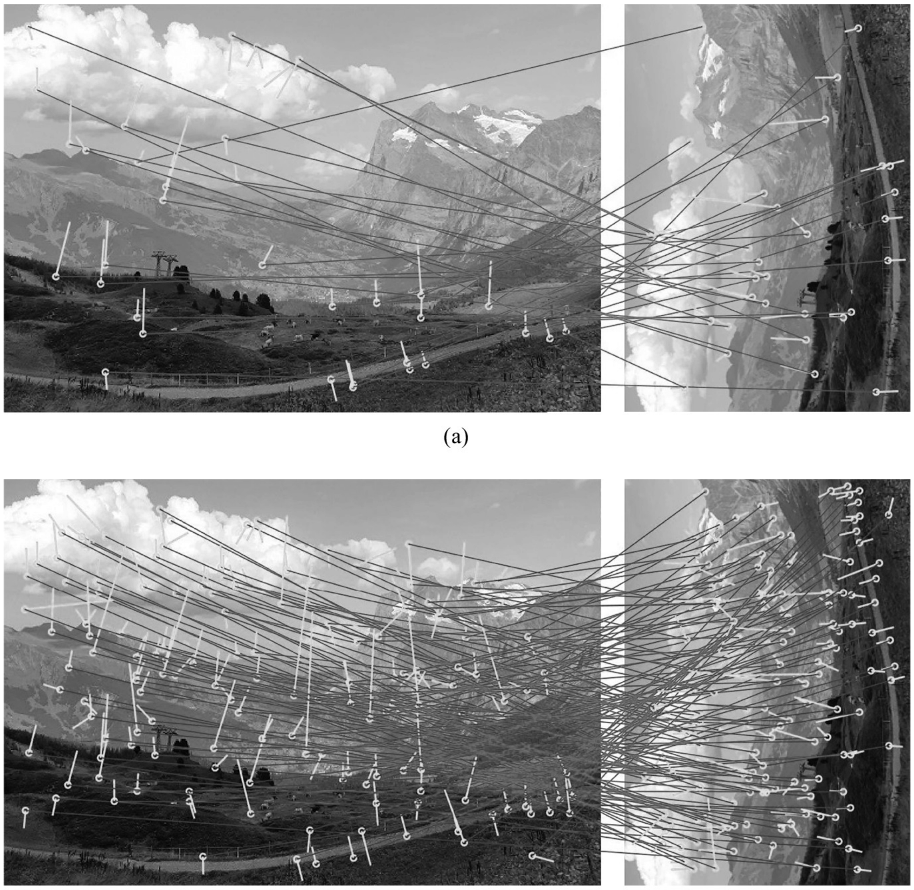
A number of points had better be further clarified before finalizing this section on scale invariant feature transform.

The technique of interpolation is not adopted for establishment of the SIFT feature point's orientation histogram , whereas it is adopted for establishment of the sample blocks' orientation histograms that form the SIFT feature point's local image descriptor. Why cannot the technique of interpolation be adopted for both steps? In fact, the technique of interpolation can be adopted for establishment of the SIFT feature point's orientation histogram , but *unnecessary*, because in this step, there are dense enough bins (conventionally 36 bins) in the SIFT feature point's orientation histogram and the histogram bins themselves can fairly represent all orientations within $[0, 2\pi)$.

Then why cannot one also set more bins in the sample blocks' orientation histograms and hence save the usage of interpolation? In fact, one can increase the number of bins in the sample blocks' orientation histograms but *still needs interpolation*. The reason is that without interpolation, a small error in the SIFT feature point's orientation tends to result in abrupt changes of the sample blocks' orientation histograms and hence make the SIFT feature point's local image descriptor unstable and susceptible to noises. Since interpolation is after all needed, it would be unnecessary to simply increase the number of bins in each sample block's orientation histogram , when a moderate number of bins (conventionally 8 bins) can already perform well in practice.

Another point concerns the sample pixel array (i.e. the local image sample patch). In conventional implementation, the sample pixel array size is scaled adaptively according to the filtering-scale parameter , i.e. the σ parameter of the Gaussian filtered image in the octave. It is worth noting that in various practical applications such as image stitching and visual SLAM where no apparent scale changes exist among target images, the scaling factor τ^{k-1} in (2.60) can be saved.[22] Yet for general cases, the scaling factor τ^{k-1} in (2.60) is indeed indispensable and can by no means be saved. For example, two images of the same scenario are illustrated in Fig. 2.30, where the smaller image is a rotated and scaled version of the larger one. The scale differs by a factor of about one and a half (not an exponential of two) among the two images. Figure 2.30a demonstrates matching effect based on SIFT feature points extracted without using the scaling factor τ^{k-1} in (2.60), whereas Fig. 2.30b demonstrates matching effect based on SIFT feature points extracted using the scaling factor τ^{k-1} in (2.60). SIFT feature points extracted in the conventional way perform desirably but those extracted without using the scaling factor perform poorly. The comparison well reflects utility of the scaling factor τ^{k-1} in (2.60) for local image descriptor generation in general cases of practical applications—Even in cases where the scaling factor τ^{k-1} in (2.60) do can be saved, yet it is unnecessary to do so. After all, keeping the scaling factor τ^{k-1} in (2.60) will incur negligible increase of computational consumption to execution of (2.60).

The last point to mention is that strictly speaking, validity of the matching method previously described in Sect. 2.6.3 is based on two assumptions: first, corresponding SIFT feature points have almost the same local image descriptor; second, non-corresponding SIFT feature points have apparently distinct local image descriptors. In practical applications, the first assumption normally holds, whereas the second assumption may be incidentally violated for few non-corresponding SIFT feature points. Consequently, by the matching method, wrong correspondence may indeed be established for few non-corresponding SIFT feature points, though absolutely most part of corresponding SIFT feature points can be correctly put into correspondence. Fortunately, this is usually of no problem to practical applications, because SIFT feature points extracted from different images are normally registered or matched on the whole, which can be effectively handled despite existence of a small part of correspondence outliers. Presentation of relevant knowledge will be postponed to Chap. 6.



(a)

(b)

Fig. 2.30 Utility of the scaling factor for local image descriptor generation **a** matching effect based on SIFT feature points extracted without using the scaling factor; **b** matching effect based on SIFT feature points extracted using the scaling factor

2.7 Post-SIFT Features

The *scale invariant feature transform* (SIFT) is a milestone glory of research in computer vision, which is attributed not uniquely to Lowe who “stood on shoulders of giants” and coined the term SIFT in [5], but as well to all pioneer researchers who built many valuable theory and methodology bricks step by step until the debut of the SIFT in the kingdom of computer vision.

The SIFT has ever since been one of the most influential features in computer vision, because simply speaking, it is indeed generally applicable, effective, and robust. Even nowadays, i.e. almost two decades after its debut, the SIFT is still not outdated but vigorous for practical applications [10–13]. Besides, the SIFT, as a comprehensive embodiment of various general methodology spirits contributed by pioneer researchers (including Lowe) since 1970s, has also been serving as “shoulders of giants” and inspiring many research works that still flourish nowadays, be the inspired research works directly SIFT-based [14–17] or not.

As readers can see from previous long presentation of knowledge on the SIFT, the success of the SIFT is not out of accidental discovery via empirical trials-and-errors, but is deeply rooted in solid theoretical analysis. The SIFT is somehow like a “theoretically derived” feature extractor instead of certain *ad hoc* one figured out empirically: The SIFT is indeed complicated and involves many aspects of details, yet almost all the details have reasonable theoretical bases—The SIFT is also a representative example reflecting power of (mathematics) theory for practical applications.

The SIFT is so influential that post-SIFT researches on feature extraction (including feature detection or interest point detection, and feature description) tend to target the SIFT explicitly or implicitly. As mentioned, the SIFT is generally applicable; in fact, it aims at general feature extraction while relying only on the most basic a priori knowledge and assumptions that are commonly shared by relevant researches. For special applications where more *ad hoc* knowledge and assumptions can be adopted and exploited explicitly or implicitly, it would be possible and even likely to figure out certain new feature extractors with desirable performance and to demonstrate certain marginal advantage of them over the SIFT in terms of feature quality. Yet on the whole, quality of SIFT features has rarely been criticized but normally admired by post-SIFT researches on feature extraction and even by among them the ones that report new feature extractors with certain marginal advantage over the SIFT in terms of feature quality.

Post-SIFT researches on feature extraction usually target the SIFT for the point that it is computationally not so efficient. On one hand, it is worth noting that one had better not exaggerate the so-called computational inefficiency of the SIFT. In fact, the SIFT is rather computational efficient for a large variety of practical applications, even at the time of its debut (not to mention nowadays when computational devices are much more powerful); otherwise it would not achieve the success as it has achieved. The SIFT turns to be computationally inefficient only for highly real-time applications such as autonomous navigation involved in mobile robotics [18–20]. Besides, as computational devices will become even more and more powerful according to the *Moore law*, current shortcoming of the SIFT in terms of computational inefficiency for highly real-time applications tends to gradually fade away.

On the other hand, it is after all desirable to have feature extractors that would perform roughly as well as the SIFT in terms of feature quality, at least for *ad hoc* applications, but incur much less computational loads. Such “much-faster-roughly-good” feature extractors will still be of value even when the SIFT can satisfy requirements of highly real-time applications one day, because much less computational loads imply much less consumption of energy, which can more or less contribute to the development of a “low-carbon” and “energy-saving” society. It is right with the motivation of developing “much-faster-roughly-good” feature extractors that relevant post-SIFT researches are conducted, usually along several directions that correspond to various methodology aspects of the SIFT respectively.

2.7.1 Fast Filtering: Example of SURF

As clarified previously during presentation of the SIFT, Gaussian filtering plays an indispensable role in scale-space analysis which provides a powerful methodology for feature extraction. So Gaussian filtering plays an indispensable role as well for the SIFT and other kinds of feature extractors that take advantage of this powerful methodology. Although the computational load of Gaussian filtering is not unaffordable for a large variety of practical applications, it is after all a factor that restricts computational efficiency of scale-space feature extraction, even when the technique of *Gaussian pyramid* is adopted in the SIFT to mitigate the problem.

For fast filtering dedicated to scale-space feature extraction, an idea is to adopt a more efficient implementation of Gaussian filtering [21, 22] in feature extraction, yet computational efficiency enhancement brought by this idea is rather limited, because any more efficient implementation of Gaussian filtering is after all an implementation of Gaussian filtering and certain minimum computational load determined inherently by Gaussian filtering is inevitable.

Another idea is to approximate effects of Gaussian filtering by using computationally much more efficient filtering methods (usually at the price of somewhat sacrificing feature quality). The **speeded up robust features (SURF)** [23] is a representative feature extractor that resorts to such idea. The SURF takes advantage of the integral image technique (see Appendix A) for fast implementation of box filtering that approximates the Hessian matrix of the Gaussian filter (kernel). The trace of the Hessian matrix is right the Laplacian of the Gaussian filter, which is approximately computed in the SIFT.

Given the parametric Gaussian filter $g(u, v, \sigma)$ defined in (2.32), compute the Hessian matrix of the Gaussian filter (kernel):

$$H(u, v, \sigma) = \begin{bmatrix} g_{uu} & g_{uv} \\ g_{uv} & g_{vv} \end{bmatrix}, \tag{2.63}$$

where

$$\begin{aligned} g_{uu} \equiv \quad & \frac{\partial^2 g}{\partial u^2} = \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{u^2}{\sigma^6} - \frac{1}{\sigma^4} \right), \\ g_{uv} \equiv \quad & \frac{\partial^2 g}{\partial u \partial v} = \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \frac{uv}{\sigma^6}, \\ g_{vv} \equiv \quad & \frac{\partial^2 g}{\partial v^2} = \frac{1}{2\pi} \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \left(\frac{v^2}{\sigma^6} - \frac{1}{\sigma^4} \right). \end{aligned}$$

The second-order Gaussian derivatives g_{uu}, g_{uv}, and g_{vv} discretized over the 9×9 mask (the initial scale in the SURF) are visualized respectively in Fig. 2.31a–c. The SURF approximates the second-order Gaussian derivatives g_{uu}, g_{uv}, and g_{vv} via box filtering masks that are visualized in Fig. 2.31d–f.

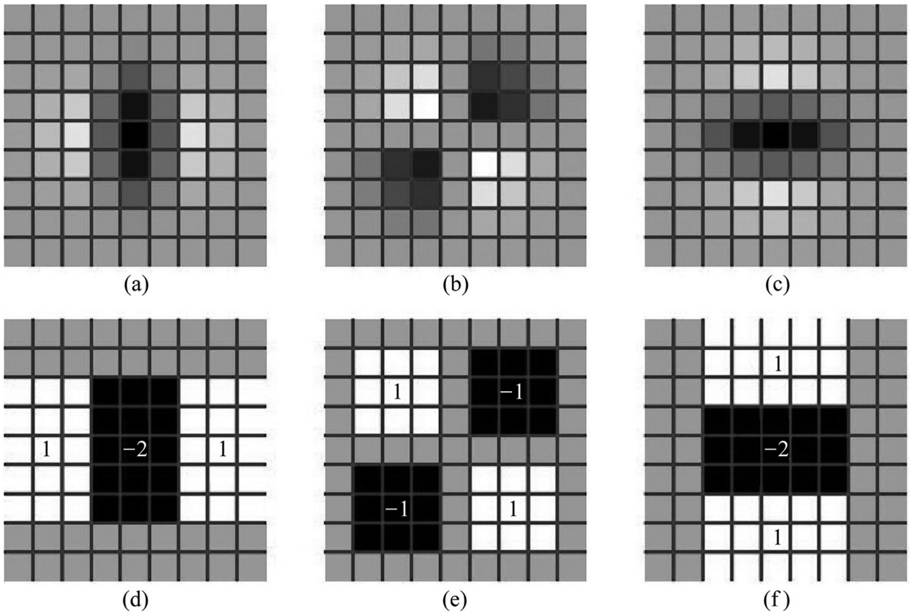


Fig. 2.31 Approximation of Gaussian derivatives ($\sigma = 1.2$) **a**–**c** discretized second-order Gaussian derivatives g_{uu}, g_{uv}, and g_{vv}; **d**–**f** Box filtering masks approximating g_{uu}, g_{uv}, and g_{vv} respectively

As explained in Appendix A, the integral image can be computed just by one traversal of the original image, according to the recursive relationship described in (A.3). Once the integral image is established, the integral image over any rectangular image are can be computed efficiently via (A.4), at a computational complexity of only $O(1)$. Therefore, box filtering involved in the SURF can be realized efficiently as well. Besides, by taking advantage of the integral image, the SURF can perform multi-scale box filtering directly on the original image, instead of relying on an extra structure like the Gaussian pyramid which has to be constructed iteratively.

After fast approximation of the second-order Gaussian derivatives g_{uu}, g_{uv}, and g_{vv}, the approximations and the associated Hessian matrix of which are denoted as

$$\bar{H}(u, v, \sigma) = \begin{bmatrix} \bar{g}_{uu} & \bar{g}_{uv} \\ \bar{g}_{uv} & \bar{g}_{vv} \end{bmatrix},$$

the SURF computes a quasi-determinant of the Hessian matrix

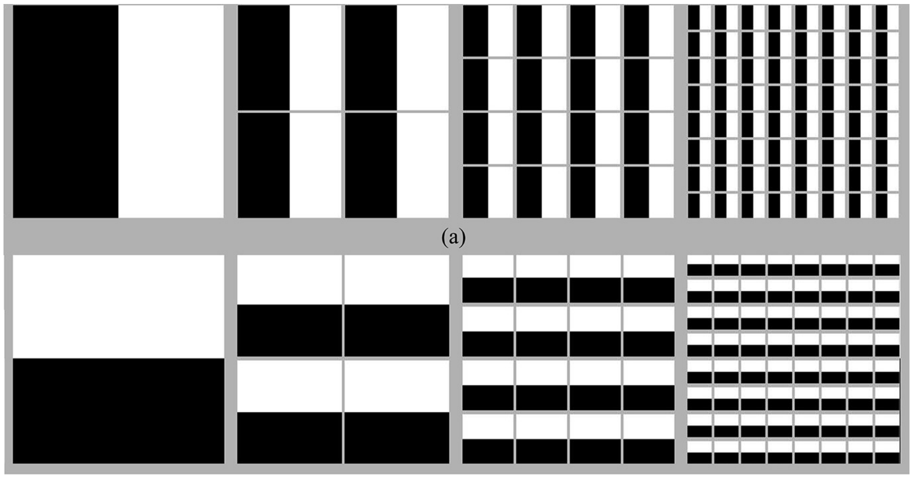
$$\det{}_{\text{quasi}}\left(\bar{H}(u, v, \sigma)\right) = \bar{g}_{uu}\bar{g}_{vv} - (0.9\bar{g}_{uv})^2 \tag{2.64}$$

which is normalized with respect to the mask size. Then the SURF resorts to the Hessian quasi-determinant specified in (2.64) for feature point detection over box filtering responses at multiple scales. Three-dimensional non-maximum suppression and maxima interpolation are also involved in the SURF for refinement of detected feature points, as in the SIFT.

It is worth noting that the integral image involved in fast approximation of Gaussian derivatives via box filtering is also used to efficiently compute *Haar-wavelet* responses [24, 25] for SURF orientation assignment and SURF description. The Haar-wavelet responses are obtained by convolving the image with the Haar basis

$$
\begin{aligned}
h_0(x) \equiv{} & \begin{cases} 1, & x \in [0,1) \\ 0, & \text{otherwise} \end{cases} \\
h_1(x) = h_{2^0+0}(x) ={} & h(x) \equiv \begin{cases} 1, & x \in [0,1/2) \\ -1, & x \in [1/2,1) \\ 0, & \text{otherwise} \end{cases} \\
h_2(x) = h_{2^1+0}(x) ={} & 2^{1/2} h(2^1 x - 0) = \sqrt{2} h(2x) \\
h_3(x) = h_{2^1+1}(x) ={} & 2^{1/2} h(2^1 x - 1) = \sqrt{2} h(2x - 1) \\
h_4(x) = h_{2^2+0}(x) ={} & 2^{2/2} h(2^2 x - 0) = 2h(4x) \\
h_5(x) = h_{2^2+1}(x) ={} & 2^{2/2} h(2^2 x - 1) = 2h(4x - 1) \\
h_6(x) = h_{2^2+2}(x) ={} & 2^{2/2} h(2^2 x - 2) = 2h(4x - 2) \\
h_7(x) = h_{2^2+3}(x) ={} & 2^{2/2} h(2^2 x - 3) = 2h(4x - 3) \\
\cdots \qquad & \cdots \\
h_{2^j+k}(x) ={} & 2^{j/2} h(2^j x - k) \quad j \geqslant 0, 0 \leqslant k < 2^j \\
\cdots \qquad & \cdots
\end{aligned}
$$

The first several Haar kernels extended for two-dimensional processing are visualized in Fig. 2.32 as example.


(a)

(b)

Fig. 2.32 Haar kernels for two-dimensional processing **a** along the *u*-axis; **b** along the *v*-axis

2.7.2 Fast Feature Detection: Example of FAST

Another direction along which relevant post-SIFT researches are conducted is to design fast feature detectors, considering the fact that the feature detection part in the SIFT does incur a considerable percentage of the overall computational load of the SIFT. The **features from accelerated segment test (FAST)** [26, 27] is a representative method for fast feature detection.

The FAST aims at fast detection of corner points. As presented in Sect. 2.4, the Harris corner detection [4] method is a classical method of corner detection and has rather desirable performance in terms of corner feature quality. On the other hand, the three rounds of Gaussian filtering over the three auxiliary images I_{uu}, I_{uv}, and I_{vv} involved in Harris corner detection tend to incur unnecessary computations that are “wasted” on obviously non-corner points. If a fast preliminary checking that can sift out obviously non-corner points is available and hence relevant computations especially Gaussian filtering can be reserved only for sparse candidate points that are likely to be corners, then computational efficiency of corner detection can be significantly enhanced. The FAST follows this idea and resorts to a preliminary checking of each pixel value against its surrounding pixel values.

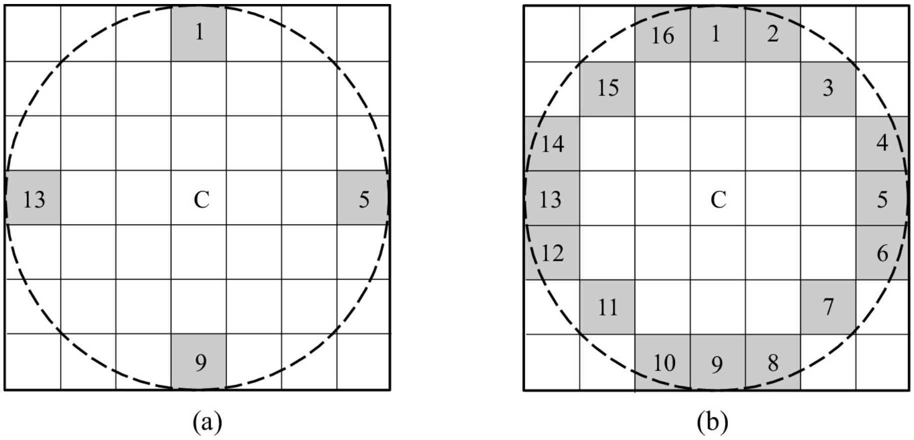


Fig. 2.33 Circular mask for fast preliminary checking **a** 4-position partial checking; **b** 16-position full checking

As illustrated in Fig. 2.33, a generic pixel is indicated by the central pixel C and its surrounding pixels for preliminary corner checking are indicated by the discrete circular mask, i.e. pixels at position 1 to position 16. Given certain threshold *t*, the pixel C passes the preliminary corner checking if its pixel value is larger by *t* or smaller by *t* than at least twelve contiguous pixel values located within the sixteen positions of the circular mask.

In fact, for each pixel, it is unnecessary to perform the 16-position full checking immediately at the very beginning. Instead, the 4-position partial checking illustrated in Fig. 2.33a can be performed first. More specifically, the pixel C passes the 4-position partial checking if its pixel value is larger by *t* or smaller by *t* than at least three pixel values located within the four positions 1, 5, 9, and 13. The 16-position full checking is performed for the pixel C only after it passes the 4-position partial checking. In practice, the 4-position partial checking, which costs little computationally, is already effective to sift out absolutely most non-corner points and leaves only a small percentage of points further for the 16-position full checking.

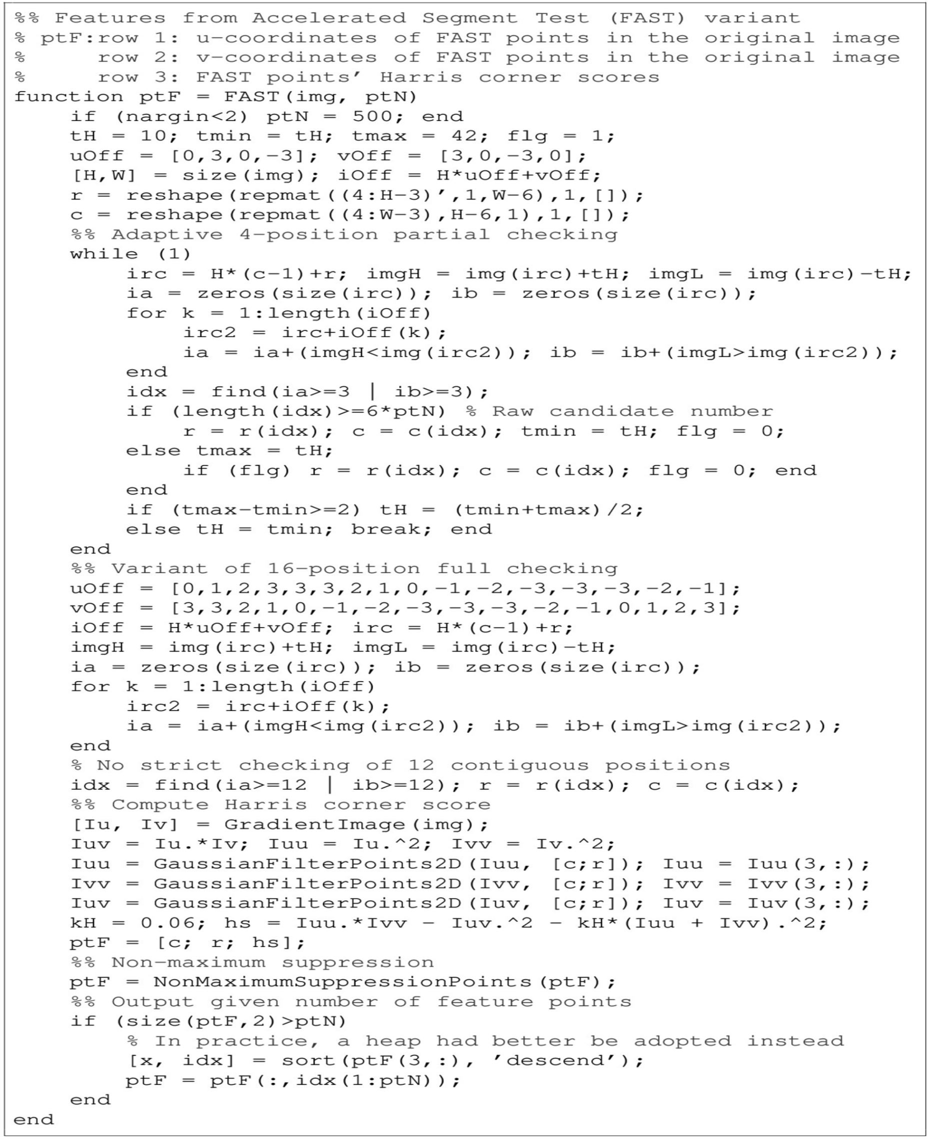
A naive implementation of the 16-position full checking is obviously unacceptable, but an algorithmically well designed implementation of the 16-position full checking can be efficient. Despite its already satisfactory computational efficiency, one may adopt a simplified variant of the 16-position full checking, which enables even more efficient implementation of checking but sacrifices checking effect little in practice. The variant of the 16-position full checking is that the pixel C passes the checking if its pixel value is larger by *t* or smaller by *t* than at least twelve pixel values located within the sixteen positions of the circular mask. In other words, the “contiguous” requirement is removed from the original 16-position full checking.

To control the number of candidate corner points output by the preliminary corner checking module, one can augment the 4-position partial checking module by an adaptive mechanism in the spirit of binary search. Simply speaking, if the output number is too large, increase the threshold *t*; if the output number is too small, decrease the threshold *t*.

Once a sparse set of candidate corner points is obtained after fast preliminary corner checking, relevant Gaussian filtering, non-maximum suppression, and Harris corner score computation are performed only for the sparse set instead of the dense set of entire image pixels, and hence a large part of computations incurred in the Harris corner detector can be saved. This accounts for the significant advantage of the FAST over the Harris corner detector in terms of computational efficiency.

Matlab code for the FAST that adopts the 4-position partial checking with the adaptive mechanism and adopts the variant of the 16-position full checking is given as follows. The involved **GaussianFilterPoints2D** function and **NonMaximumSuppressionPoints** function are given in Sect. C.4.2 in Appendix C and in Sect. 2.2.4 respectively.

FAST.m



```
%% Features from Accelerated Segment Test (FAST) variant
% ptF:row 1: u-coordinates of FAST points in the original image
%     row 2: v-coordinates of FAST points in the original image
%     row 3: FAST points’ Harris corner scores
function ptF = FAST(img, ptN)
    if (nargin<2) ptN = 500; end
    tH = 10; tmin = tH; tmax = 42; flg = 1;
    uOff = [0,3,0,-3]; vOff = [3,0,-3,0];
    [H,W] = size(img); iOff = H*uOff+vOff;
    r = reshape(repmat((4:H-3)’,1,W-6),1,[]);
    c = reshape(repmat((4:W-3),H-6,1),1,[]);
    %% Adaptive 4-position partial checking
    while (1)
        irc = H*(c-1)+r; imgH = img(irc)+tH; imgL = img(irc)-tH;
        ia = zeros(size(irc)); ib = zeros(size(irc));
        for k = 1:length(iOff)
            irc2 = irc+iOff(k);
            ia = ia+(imgH<img(irc2)); ib = ib+(imgL>img(irc2));
        end
        idx = find(ia>=3 | ib>=3);
        if (length(idx)>=6*ptN) % Raw candidate number
            r = r(idx); c = c(idx); tmin = tH; flg = 0;
        else tmax = tH;
            if (flg) r = r(idx); c = c(idx); flg = 0; end
        end
        if (tmax-tmin>=2) tH = (tmin+tmax)/2;
        else tH = tmin; break; end
    end
    %% Variant of 16-position full checking
    uOff = [0,1,2,3,3,3,2,1,0,-1,-2,-3,-3,-3,-2,-1];
    vOff = [3,3,2,1,0,-1,-2,-3,-3,-3,-2,-1,0,1,2,3];
    iOff = H*uOff+vOff; irc = H*(c-1)+r;
    imgH = img(irc)+tH; imgL = img(irc)-tH;
    ia = zeros(size(irc)); ib = zeros(size(irc));
    for k = 1:length(iOff)
        irc2 = irc+iOff(k);
        ia = ia+(imgH<img(irc2)); ib = ib+(imgL>img(irc2));
    end
    % No strict checking of 12 contiguous positions
    idx = find(ia>=12 | ib>=12); r = r(idx); c = c(idx);
    %% Compute Harris corner score
    [Iu, Iv] = GradientImage(img);
    Iuv = Iu.*Iv; Iuu = Iu.^2; Ivv = Iv.^2;
    Iuu = GaussianFilterPoints2D(Iuu, [c;r]); Iuu = Iuu(3,:);
    Ivv = GaussianFilterPoints2D(Ivv, [c;r]); Ivv = Ivv(3,:);
    Iuv = GaussianFilterPoints2D(Iuv, [c;r]); Iuv = Iuv(3,:);
    kH = 0.06; hs = Iuu.*Ivv - Iuv.^2 - kH*(Iuu + Ivv).^2;
    ptF = [c; r; hs];
    %% Non-maximum suppression
    ptF = NonMaximumSuppressionPoints(ptF);
    %% Output given number of feature points
    if (size(ptF,2)>ptN)
        % In practice, a heap had better be adopted instead
        [x, idx] = sort(ptF(3,:), ’descend’);
        ptF = ptF(:,idx(1:ptN));
    end
end
```

The sparse set of candidate corner points are sorted in descending order according to their Harris corner scores and only the first certain predefined number of corner points are output by the FAST. In practice, a complete sorting of the candidate corner points is unnecessary, whereas a heap data structure can be utilized for the needed partial sorting.

2.7.3 Simplified Feature Description: Example of BRIEF

The third direction for relevant post-SIFT researches is to simplify feature descriptors yet without sacrificing much their ability to represent and distinguish features. Simplification of feature descriptors is not only meaningful in the sense that feature descriptors themselves can be obtained more efficiently but also meaningful in the sense that they tend to be matched more efficiently as well. The **binary robust independent**

elementary features (BRIEF) [28] is a representative method to obtain simplified feature descriptors.

The BRIEF resorts to binary feature representation. Each bit of the BRIEF descriptor vector corresponds to the pixel value comparison test between a specific pair of locations in the local patch of the feature point. Given N_{BRIEF} such local pairs of locations

$$\boldsymbol{S}_{\mathrm{BRIEF}} = \left\{ \left\{ \begin{bmatrix} u_1^a \\ v_1^a \end{bmatrix}, \begin{bmatrix} u_1^b \\ v_1^b \end{bmatrix} \right\}, \left\{ \begin{bmatrix} u_2^a \\ v_2^a \end{bmatrix}, \begin{bmatrix} u_2^b \\ v_2^b \end{bmatrix} \right\}, \ldots, \left\{ \begin{bmatrix} u_{N_{\mathrm{BRIEF}}}^a \\ v_{N_{\mathrm{BRIEF}}}^a \end{bmatrix}, \begin{bmatrix} u_{N_{\mathrm{BRIEF}}}^b \\ v_{N_{\mathrm{BRIEF}}}^b \end{bmatrix} \right\} \right\}$$

for BRIEF feature description and a generic feature point located at $[u \quad v]^{\mathrm{T}}$, the feature’s BRIEF descriptor vector is formed in the binary form as

$$\boldsymbol{F}_{u,v} = \sum_{k=1}^{N_{\mathrm{BRIEF}}} 2^{k-1} \tau \left(\begin{bmatrix} u + u_k^a \\ v + v_k^a \end{bmatrix}, \begin{bmatrix} u + u_k^b \\ v + v_k^b \end{bmatrix} \right), \tag{2.65}$$

where $\tau(\cdot)$ is the pixel value comparison test function such that

$$\tau\left(\begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} \right) = \begin{cases} 1, & I(u_1, v_1) < I(u_2, v_2) \\ 0, & \text{otherwise} \end{cases}$$

The location pairs in $\boldsymbol{S}_{\mathrm{BRIEF}}$ are called *sampling patterns* for BRIEF description. A simple yet effective enough way to have such set of sampling patterns is to generate them randomly in a local patch about the origin.[23] Matlab code for random generation of sampling patterns is given as follows.

SamplingPatternsBRIEF.m

```
%% Obtain sampling patterns for BRIEF
function briefsmp = SamplingPatternsBRIEF()
    if(exist(’brief_sampling.mat’))
        load(’brief_sampling.mat’); % Get sampling patterns
    else % Generate sampling patterns if not existing
        briefsmp = zeros(256,4); fprintf(’briefsmp = [’);
        for k=1:256
            while(1) % xy = random(’norm’,0,7,1,4);
                if (min(xy)>=-13 && max(xy)<=13) break; end
            end
            briefsmp(k,:) = round(xy);
            if (k<256) fprintf(’%d, %d, %d, %d;’,briefsmp(k,:));
            else fprintf(’%d, %d, %d, %d];\n’,briefsmp(k,:));
            end
        end
        save(’brief_sampling.mat’, ’briefsmp’);
    end
end
```

If feature points have associated orientations, then (2.65) has an extended formalism for rotation-aware BRIEF descriptors

$$\boldsymbol{F}_{u,v} = \sum_{k=1}^{N_{\mathrm{BRIEF}}} 2^{k-1} \tau \left(\begin{bmatrix} u \\ v \end{bmatrix} + \boldsymbol{R}(\theta) \begin{bmatrix} u_k^a \\ v_k^a \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix} + \boldsymbol{R}(\theta) \begin{bmatrix} u_k^b \\ v_k^b \end{bmatrix} \right), \tag{2.66}$$

where θ denotes the orientation of a generic feature point and $\boldsymbol{R}(\theta)$ denotes the rotation matrix

$$\boldsymbol{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

The BRIEF descriptor vector established according to the rotation-aware formalism (2.66) enables the feature to possess rotation-invariant representation ability. Matlab code for the rotation-aware BRIEF is given as follows.

RotationBRIEF.m

```
%% Binary Robust Independent Elementary Features (BRIEF)
function fD = RotationBRIEF(img, ptF, briefsmp)
    % Four 64-bit integers represent 256 bits
    n = size(ptF,2); fD = uint64(zeros(4,n));
    [H,W]=size(img); imgD = IntegralImageFilter(img);
    for k=1:n
        u = ptF(1,k); v = ptF(2,k); a=ptF(3,k);
        rotM=[cos(a),-sin(a);sin(a),cos(a)];
        pos1 = round(rotM*briefsmp(:,1:2)');
        pos2 = round(rotM*briefsmp(:,3:4)');
        for j=1:256
            u1 = max(1,min(W,u+pos1(1,j)));
            v1 = max(1,min(H,v+pos1(2,j)));
            u2 = max(1,min(W,u+pos2(1,j)));
            v2 = max(1,min(H,v+pos2(2,j)));
            I1 = imgD(v1,u1); I2 = imgD(v2,u2);
            j1 = ceil(j/64); j2 = j-64*(j1-1);
            % Bit representation
            fD(j1,k) = bitset(fD(j1,k),j2,(I1<I2));
        end
    end
end
```

In the given implementation of the rotation-aware BRIEF, the integral image based filter, i.e. the sub-function **IntegralImageFilter.m** (see Appendix A) is adopted to enhance the stability of BRIEF feature description.

In practice, the dimension of the BRIEF descriptor vector can be flexible, yet had better be multiples of 64 (or 32) such as the demonstrated 256 to take best advantage of *bit representations* in the conventional 64-bit (or 32-bit) computer data format and of *bit operations*. Each 256-dimensional BRIEF descriptor vector is stored in four 64-bit integers. The bit-wise *XOR* (*eXclusive OR*) operation[24] is especially suitable to compute the *Hamming distance* [29] that characterizes the difference between two BRIEF descriptors.

2.7.4 Oriented FAST and Rotated BRIEF (ORB)

The **oriented FAST and rotated BRIEF (ORB)** [30] is a representative post-SIFT feature that is well-known in real-time applications, especially in mobile robotics [31–33]. The ORB takes the FAST for feature detection and takes the rotation-aware BRIEF for feature description. Concerning feature detection, the ORB augments the original FAST by attaching orientations to the detected corners. The orientation attached to a FAST feature is computed using the *intensity centroid* [34]. Define the moments (see Sect. C.1 in Appendix C) of a local patch as

$$m_{ij} = \textstyle\sum_{u,v} u^i v^j I(u,v) \tag{2.67}$$

and then compute the centroid as

$$\boldsymbol{C} = \begin{bmatrix} \frac{m_{10}}{m_{00}} & \frac{m_{01}}{m_{00}} \end{bmatrix}^T. \tag{2.68}$$

Further compute the orientation of the local patch as

$$\theta = \text{arctan}\ \frac{m_{01}}{m_{10}} = \frac{\sum\limits_{u,v} vI(u,v)}{\sum\limits_{u,v} uI(u,v)} \qquad (\text{quadrant} - \text{aware arctan}) \tag{2.69}$$

Matlab code for computing the feature orientation is given as follows.

IntensityCentroid.m

```
%% Compute corner orientations via intensity centroid
function ang = IntensityCentroid(img, pt)
    uOff = repmat((-3:3),7,1); vOff = repmat((-3:3)’,1,7);
    ang = zeros(1,size(pt,2)); imgD = double(img);
    for k=1:size(pt,2)
        u = pt(1,k); v = pt(2,k);
        m10 = sum(sum(uOff.*imgD(v-3:v+3,u-3:u+3)));
        m01 = sum(sum(vOff.*imgD(v-3:v+3,u-3:u+3)));
        ang(k) = atan2(m01,m10);
    end
end
```

The ORB, also in the spirit of multi-scale processing, detects rotation-aware FAST corner points at multiple scales. More specifically, the ORB first detects rotation-aware FAST corner points from the original image. Then the ORB resizes the image by a scaling factor of $\sqrt{1/2}$ and detects rotation-aware FAST corner points from the resized image. The ORB resizes the image further by the same scaling factor and detects rotation-aware FAST corner points from the further resized image. The ORB continues such image resizing and corner detection process until a specified set of scales (for example, 1, $\sqrt{1/2}$, 1/2, $\sqrt{1/2}/2$, 1/4) are traversed. Once rotation-aware FAST corner points at multiple scales are detected, the ORB resorts to the rotation-aware BRIEF to retrieve binary (bit-represented) BRIEF descriptors associated with the previously detected FAST corner points. Matlab code for the ORB is given as follows.

ORB.m

```
%% Oriented FAST and Rotated BRIEF (ORB)
% fL: row 1: u-coordinates of ORB points in the original image
%     row 2: v-coordinates of ORB points in the original image
%     row 3: ORB points’ orientation angles in [0, 2*pi)
% fD: ORB points’ corresponding (vector) descriptors
function [fL, fD] = ORB(img, ptN)
    if (nargin<2) ptN = 500; end
    sk = [1, 0.7071, 0.5, 0.3536, 0.25]; % Multiple scales
    %% Detect ORB corners at multiple scales via FAST
    ptF = []; ang = [];
    for k=1:length(sk)
        imgk=imresize(img,sk(k)); ptNk=round(ptN*sk(k)^2);
        ptFk = FAST(imgk, ptNk);
        angk = IntensityCentroid(imgk, ptFk(1:2,:));
        ptFk(1:2,:) = round(ptFk(1:2,:)/sk(k));
        ptF = [ptF, [ptFk;angk]];
    end
    % In practice, a heap had better be adopted instead
    [x, idx] = sort(ptF(3,:), ’descend’);
    fL = ptF([1,2,4],idx(1:min(ptN,length(idx))));

    %% Get binary descriptors via rotation-aware BRIEF
    briefsmp = SamplingPatternsBRIEF();
    fD = RotationBRIEF(img, fL, briefsmp);
end
```

For demonstration of the ORB performance, still take the original image illustrated in Fig. 2.20 as example. Rotate the original image to form a rotated version of the original image, as illustrated in the right sub-figure of Fig. 2.34. ORB features are extracted from the original image and the rotated image, after which the two sets of ORB features corresponding to the two images respectively are matched with each other using the simple criterion of minimum feature descriptor distance checking. An effect of (partial) correspondences between ORB features is demonstrated in Fig. 2.34, similar to that demonstrated in Fig. 2.27. As shown in Fig. 2.34, the ORB features extracted from the two images respectively have consistent locations and rotations intuitively. Besides the “dazzling” effect demonstrated in Fig. 2.34, a single example of matched pair of ORB features is demonstrated in Fig. 2.35 to highlight clearly the location and rotation consistency.

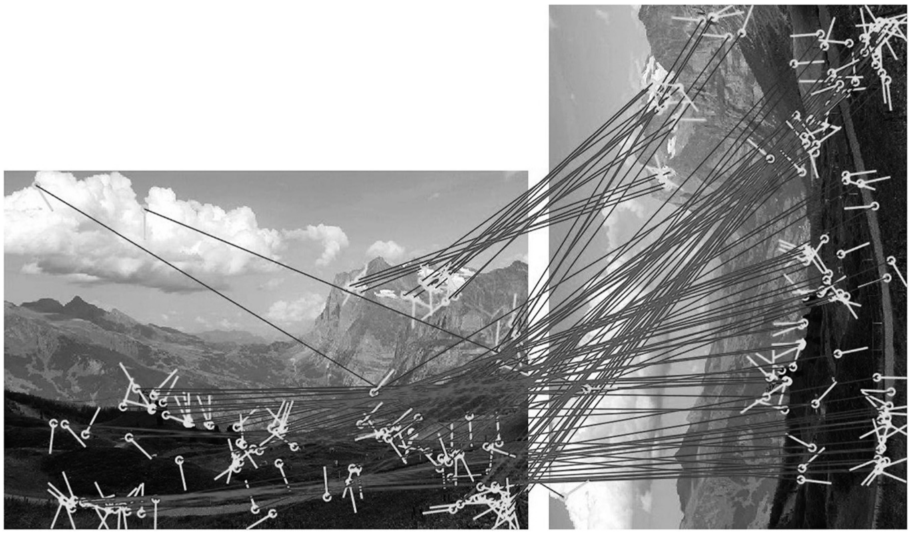


Fig. 2.34 Correspondences between ORB features extracted from the original image and the rotated image

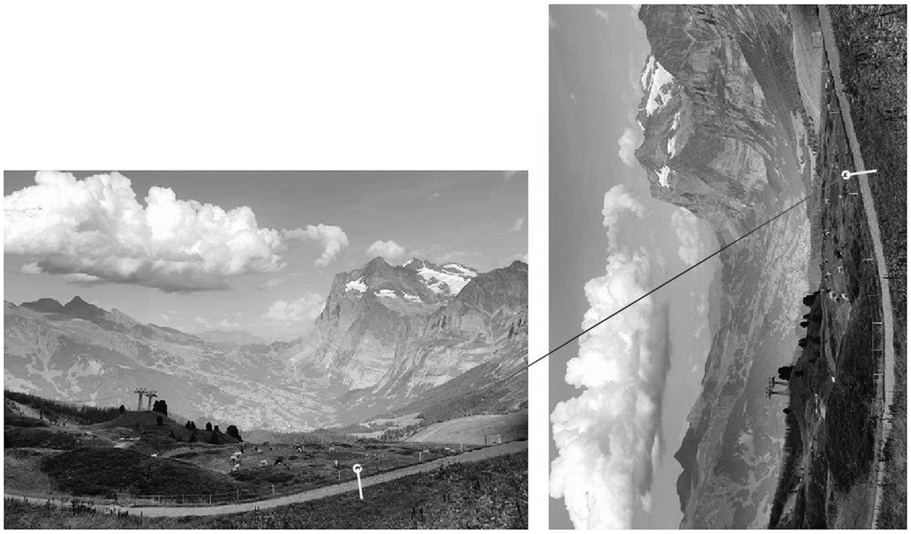
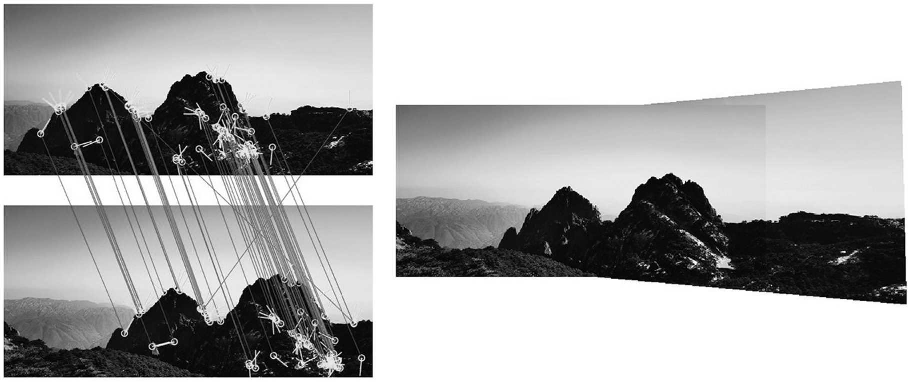


Fig. 2.35 A matched pair of ORB features between the original image and the rotated image (intuitively consistent locations and rotations)

For another demonstration example, two original images, which are captured from the same location at Huang-Shan, i.e. Yellow-Mountain but with slight viewing angle difference, are illustrated in Fig. 2.36a. Two sets of ORB features are extracted from the two images respectively and their mutual correspondences are established. On the whole, matched pairs of ORB features between the two images have consistent locations and rotations intuitively. Despite existence of few wrong correspondences, the demonstrated effectiveness of ORB based matching on the whole is already sufficient to enable realization of various interesting functions, such as image stitching [25] illustrated in Fig. 2.36b.



(a) (b)

Fig. 2.36 ORB based image stitching **a** two images and their ORB correspondences; **b** stitched image

2.8 Machine Learning Based Features

Machine learning plays an indispensable role in the kingdom of computer vision. Intensive researches have already been dedicated to machine learning since decades ago. In the last decade, supported by revolutionary progress in hardware, the renaissance of **deep learning** [35, 36] (the most popular domain in the kingdom of **machine learning** nowadays) has stimulated a powerful gush of deep learning based features or more generally, machine learning based features. The author first provides a brief overview of representative pre-deep learning methods and representative deep learning methods in general sense.

2.8.1 Pre-Deep Learning Methods

Representative pre-deep learning methods include

K-nearest Neighbours (KNN)

The **K-nearest neighbours** method [37] is a typical nonparametric classifier and is among the most popular classifiers. Several main reasons account for its popularity: First, its core idea is intuitively comprehensible and rather analogous to daily life experiences. It is like in daily life to roughly know characteristics of a person, people may usually follow a way of indirect evaluation by analysing the person's "social circle" namely people with whom the person usually get along. Similarly, to classify a feature, one may check the feature is "close" to which other features of a priori known classification results. Second, implementation of the K-nearest neighbours method is comparatively direct and simple. It is true that there are usually data structure concerns for computational efficiency, yet roughly speaking, it involves only *finding* or *searching* operations and is exempt from comparatively complicated procedures of numeric computations such as those especially for optimization frameworks. Third, the output of the K-nearest neighbours method is obviously convergent, and one does not need to worry that execution of the K-nearest neighbours method diverges or becomes oscillatory.

Various kinds of distance metrics can be adopted to characterize "closeness" (similarity) or "apartness" (distinction) among features in feature space. Knowledge on distance metrics will be postponed to Chap. 3. Besides strict distance metrics, other kinds of measures such as joint distance kernels may also be adopted in the K-nearest neighbours method, only if the adopted measures would properly serve the purpose of characterizing "closeness" or "apartness" among features.

Principal Component Analysis (PCA)

The **principal component analysis** method [38] (also known as the *Karhunen—Loève Transform* method occasionally) is probably the most popular member of the family of **dimensionality reduction** methods that include the *independent component analysis* method [39] and the *non-negative matrix factorization* method [40] as well.

The spirit of the principal component analysis method consists in computing mutually-orthogonal *principal axes* or *principal directions*, after projection on the sub-space spanned by which the variance of the given data is maximized. Given a set of generic d-dimensional feature points denoted by $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_N$. The author focuses on computing the unique principal axis $\boldsymbol{p}$ in following derivation while the idea can be naturally extended to computation of multiple principal axes. The variance of the projected feature points is

$$\begin{aligned} V(\boldsymbol{p}) \equiv \quad & E[(\boldsymbol{p}^{\mathrm{T}}\boldsymbol{x} - E[\boldsymbol{p}^{\mathrm{T}}\boldsymbol{x}])^2] = \boldsymbol{p}^{\mathrm{T}} E[(\boldsymbol{x} - E[\boldsymbol{x}])(\boldsymbol{x} - E[\boldsymbol{x}])^{\mathrm{T}}]\boldsymbol{p} \\ = \quad & \boldsymbol{p}^{\mathrm{T}} \mathrm{Cov}(\boldsymbol{x})\boldsymbol{p} \approx \boldsymbol{p}^{\mathrm{T}} \left[\frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{x}_i - \frac{1}{N}\sum_{j=1}^{N} x_j\right) \left(\boldsymbol{x}_i - \frac{1}{N}\sum_{j=1}^{N} x_j\right)^{\mathrm{T}} \right] \boldsymbol{p}. \end{aligned} \tag{2.70}$$

Suppose the principal axis $\boldsymbol{p}$ is normalized, so the problem of computing the principal axis $\boldsymbol{p}$ can be formalized as the constrained optimization problem

$$\boldsymbol{p} = \mathrm{argmax}_{\boldsymbol{p}}\ \boldsymbol{p}^{\mathrm{T}} \mathrm{Cov}(\boldsymbol{x})\boldsymbol{p} \quad \mathrm{s.t.}\ \ \boldsymbol{p}^{\mathrm{T}}\boldsymbol{p} = 1 \tag{2.71}$$

or equivalently as the unconstrained optimization problem

$$\boldsymbol{p} = \mathrm{argmax}_{\boldsymbol{p}} \ \frac{\boldsymbol{p}^{\mathrm{T}} \mathrm{Cov}(\boldsymbol{x}) \boldsymbol{p}}{\boldsymbol{p}^{\mathrm{T}} \boldsymbol{p}}. \tag{2.72}$$

The objective function in (2.72) is called the *Rayleigh ratio*. In a similar way of how to solve (F.10) in Sect. F.2 in Appendix F, one can also solve (2.72) and know that the principal axis $\boldsymbol{p}$ is right the normalized eigenvector associated with the largest eigenvalue of the covariance matrix $\mathrm{Cov}(\boldsymbol{x})$.

Support Vector Machine (SVM)

The **support vector machine** method [41] is a representative method that by no means is negligible in the kingdom of machine learning. Its importance consists mainly in two aspects: First, the support vector machine method is indeed powerful enough. In fact, if it were not because of the renaissance of deep learning, the support vector machine method especially when augmented by *kernel techniques* would still assume a role as one of dominant machine learning methods nowadays just like deep learning methods are right assuming. Even currently in the era of deep learning, the support vector machine method is still valuable, especially in applications where the amount of training data is rather limited. Second, the support vector machine method is mathematically elegant and its performance is theoretically supported. This is a merit for which even deep learning methods have been craving.

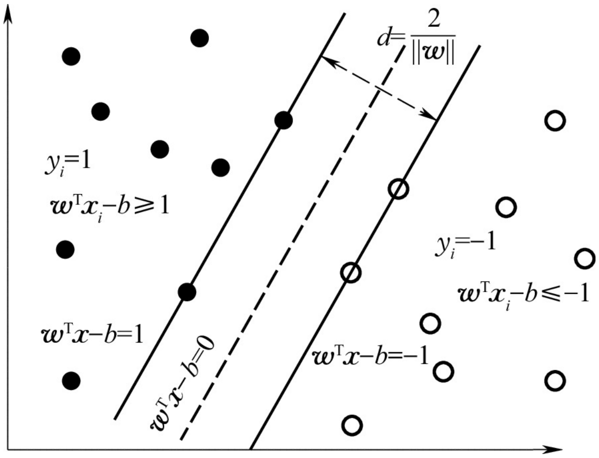

$d=\frac{2}{\|\boldsymbol{w}\|}$
$y_i=1$
$\boldsymbol{w}^{\mathrm{T}}x_i-b\geqslant 1$
$y_i=-1$
$\boldsymbol{w}^{\mathrm{T}}x_i-b\leqslant -1$
$\boldsymbol{w}^{\mathrm{T}}x-b=1$
$\boldsymbol{w}^{\mathrm{T}}x-b=0$
$\boldsymbol{w}^{\mathrm{T}}x-b=-1$

Fig. 2.37 Support vector machine: hard margin

The support vector machine method is of strong flavour of geometry. Its spirit can be naturally clarified in geometric terms, namely to use a *hyperplane* to separate the given set of feature points as wide as possible,[26] as illustrated in Fig. 2.37. Given a set of generic d-dimensional feature points denoted by $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_N$ with corresponding bi-classification labels y_1, y_2, ..., $y_N \in \{+1, -1\}$. The core problem of the support vector machine method can be formalized as the constrained optimization problem

$$\begin{aligned} & \{\boldsymbol{w}, b\} = \mathrm{argmax}_{\{\boldsymbol{w},b\}} \ \frac{2}{\|\boldsymbol{w}\|_2} \quad \mathrm{s.t.} \ \ y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b) \geqslant 1 \ \text{ for } \ i \in \{1, \cdots, N\} \\ \Leftrightarrow \quad & \{\boldsymbol{w}, b\} = \mathrm{argmin}_{\{\boldsymbol{w},b\}} \ \|\boldsymbol{w}\|_2 \quad \mathrm{s.t.} \ \ y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b) \geqslant 1 \ \text{ for } \ i \in \{1, \cdots, N\} \end{aligned} \tag{2.73}$$

where the equations $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b = 1$ and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b = -1$ represent the two boundary hyperplanes of the widest hyperplane zone that separates the two classes of feature points, and the equation $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b = 0$ represents the bisection hyperplane of the hyperplane zone, as illustrated in Fig. 2.37. The width of the hyperplane zone is $2/\|\boldsymbol{w}\|$, so the objective of (2.73) is to maximize the width of the hyperplane zone, i.e. as just mentioned above, to separate the given set of feature points as wide as possible.

When the given set of feature points are separable namely when a hyperplane zone of *hard margin* can separate the feature points distinctly into two classes, the widest hyperplane zone that separates the two classes of feature points does exist and corresponds to the solution of the constrained optimization problem (2.73). This is the case illustrated in Fig. 2.37.

As illustrated in Fig. 2.38, when the given set of feature points are nonseparable namely no hyperplane can separate the feature points distinctly into two classes. In mathematical terms, no solution of the hyperplane parameter set $\{\boldsymbol{w}, b\}$ can satisfy all the constraints specified in the constrained optimization problem (2.73), not to say the optimal solution. For such case of nonseparable feature points, instead of establishing a hyperplane zone of hard margin that separates the feature points distinctly into two classes, one may establish a hyperplane zone of *soft margin* that aims at separating the given set of feature points as wide as possible on one hand and causing as little misclassification error as possible on the other hand.

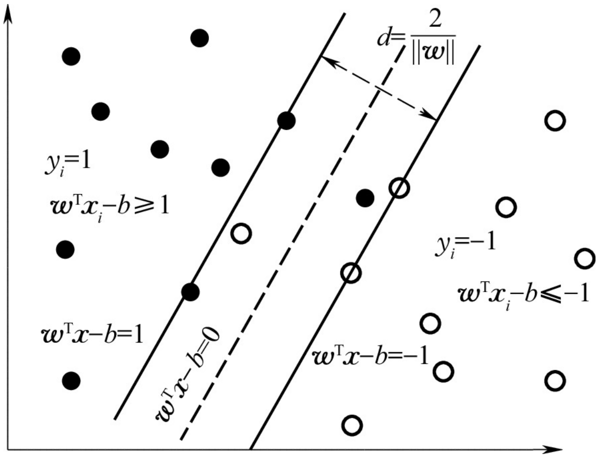

$d=\frac{2}{\|\boldsymbol{w}\|}$
$y_i=1$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i-b\geqslant 1$
$y_i=-1$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i-b\leqslant -1$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-b=1$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-b=0$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-b=-1$

Fig. 2.38 Support vector machine: soft margin

The objective of maximizing the width of the hyperplane zone of soft margin and the objective of minimizing misclassification error usually contradict each other in practical applications. A loss function that reflects the two objectives jointly can be optimized and a balance between the two objectives is usually involved. Since no solution of $\{\boldsymbol{w}, b\}$ can satisfy all the constraints specified in (2.73), it is meaningless to still treat them as hard constraints for optimization. Instead, the hard constraints are replaced by soft errors

$$e_i = \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b)\right\} = \begin{cases} 0, & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b) \geqslant 1 \\ 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b), & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b) < 1 \end{cases} \tag{2.74}$$

for $i \in \{1, \cdots, N\}$. What (2.74) conveys is that if the feature point $\boldsymbol{x}_i$ is classified correctly, then no classification error will be incurred, i.e. $e_i = 0$. Otherwise, if the feature point $\boldsymbol{x}_i$ is misclassified, then a classification error $e_i > 0$ will be incurred depending on how wrong the misclassification is.

The optimal hyperplane zone of soft margin may be obtained by solving the following unconstrained optimization problem that incorporate soft errors (2.74)

$$\{\boldsymbol{w}, b\} = \operatorname{argmin}_{\{\boldsymbol{w}, b\}} \lambda\|\boldsymbol{w}\|_2^2 + \tfrac{1}{N}\textstyle\sum_{i=1}^{N} e_i. \tag{2.75}$$

Although (2.75) is an unconstrained optimization problem, solving it is not necessarily more convenient than solving constrained optimization problems, because soft error terms in (2.75) are nonlinear and piece-wise defined. To handle (2.75), one can transform it equivalently into the constrained optimization problem

$$\begin{array}{rl} \{\boldsymbol{w}, b, \boldsymbol{e}\} = & \operatorname{argmin}_{\{\boldsymbol{w}, b, \boldsymbol{e}\}} \lambda \|\boldsymbol{w}\|_2^2 + \frac{1}{N} \sum_{i=1}^{N} e_i \\ \text{s. t.} & y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i - b) + e_i - 1 \geqslant 0 \\ & e_i \geqslant 0 \quad \text{for } i \in \{1, \cdots, N\} \end{array} \tag{2.76}$$

or compactly as

$$\begin{array}{rl} \{\boldsymbol{w}, b, \boldsymbol{e}\} = & \operatorname{argmin}_{\{\boldsymbol{w}, b, \boldsymbol{e}\}} \lambda \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} + \frac{1}{N} \mathbf{1}^{\mathrm{T}} \boldsymbol{e} \\ \text{s. t.} & \begin{bmatrix} \operatorname{diag}(\boldsymbol{Y}) \boldsymbol{X} & -\boldsymbol{Y} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{e} \end{bmatrix} \geqslant \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \end{array}$$

where

$$\begin{aligned} \boldsymbol{e} &\equiv [e_1 \quad e_2 \quad \cdots \quad e_N]^{\mathrm{T}}, \\ \mathbf{1} &\equiv [1 \quad 1 \quad \cdots \quad 1]^{\mathrm{T}}, \\ \boldsymbol{X} &\equiv [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_N]^{\mathrm{T}}, \\ \boldsymbol{Y} &\equiv [y_1 \quad y_2 \quad \cdots \quad y_N]^{\mathrm{T}}. \end{aligned}$$

Note that **1** is the all-element-one vector instead of the identity matrix **I**.

The constrained optimization problem (2.76) is in a form of *quadratic programming* [42]. Methods for solving quadratic programming are rather mature. First, use the *simplex method* [43] to solve the simplified *linear programming* version of (2.76)

$$\begin{array}{rl} \{\boldsymbol{w}, b, \boldsymbol{e}\} = & \operatorname{argmin}_{\{\boldsymbol{w}, b, \boldsymbol{e}\}} \mathbf{1}^{\mathrm{T}} \boldsymbol{e} \\ \text{s. t.} & \begin{bmatrix} \operatorname{diag}(\boldsymbol{Y}) \boldsymbol{X} & -\boldsymbol{Y} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{e} \end{bmatrix} \geqslant \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \end{array} \tag{2.77}$$

Then use the solution of (2.77) as an initial solution estimate of the original quadratic programming problem (2.76) and solve it.

Systematic theoretic analysis and kernel techniques of the support vector machine method are saved. Interested readers may refer to [41] for more details.

Adaboost

The family of **boosting** methods [44, 45] especially the **Adaboost** (namely **adaptive boosting**) method [46] also play an important role in machine learning. Their common spirit consists in "boosting" a group of already available learners to transform together into a new learner that is (much) better than each individual of them. Take the Adaboost method as example, it has demonstrated in practical applications an amazing power that it can boost a large group of weak learners [27] into a strong learner of competitive performance, achieving success especially in practice of *face detection* [48, 49] which has already been incorporated into commercial mobile devices such as cameras and smart phones for many years.

2.8.2 Deep Learning Methods

Representative deep learning methods (or deep neural network architectures) include

Convolutional Neural Networks (CNN)

Pioneer researchers have been making core contributions to the domain of artificial neural networks based learning since more than half a century ago [50–53]. See Appendix G for a brief review of theoretical fundamentals of artificial neural networks. Despite valuable pioneer research works on artificial neural networks in early years, what actually makes deep learning debut in a large variety of practical applications and popularizes deep learning in public is attributed to the **convolutional neural network** method.

Essential ideas underlying convolutional neural networks can date back to late 1980s [54], which were out of a motivation for building a reduced size artificial neural network on one hand and another motivation for incorporating field knowledge concerning computer vision on the other hand. The essential ideas include

- *Local connection* also known as *sparse interaction*: Instead of connecting with all units at the previous layer, a unit (usually a hidden unit) connects only with a local patch of units at the previous layer. The idea is motivated by conventional practices of local feature extraction for computer vision. The mechanism of local connection, which enables sparse interaction among units in artificial neural networks instead of dense interaction among units in original artificial neural networks, largely reduces both computational consumption and storage consumption involved in artificial neural network applications.
- *Parameter sharing* also known as *weight sharing*: Units at the same layer share the same set of parameters or weights for their respective connections with local patches of units at the previous layer. The mechanism of parameter sharing largely reduces storage consumption involved in artificial neural network applications. Since units at the same layer share the same set of connection weights, it is like they accumulate their input signals by convolving unit output signals of the previous layer with a kernel determined by the shared configuration of local connections. This is why convolutional neural networks are called “convolutional” neural networks.
- *Partial parameter sharing*: Units at the same layer do not have all their parameters shared as a common set of parameters. They have independent biases respectively. The fact that units at the same layer share the same set of local connection weights but have independent biases enables shift-invariant or translation-invariant extraction of lower-level features on one hand and still maintains availability of approximate spatial information for extraction of higher-level features on the other hand. Otherwise, if units at the same layer even share the same bias, then spatial information will be totally lost.
- *Pooling*: This is to gather statistics of a local patch of lower-layer units into a higher-layer unit. One may make analogy between pooling in the convolutional neural network and down-sampling in the Gaussian pyramid based SIFT extractor, both of which are motivated by the multi-scale adaptive perception ability of humans’ biological vision systems. As perception and cognition of features augment to a larger scale, humans’ biological vision systems tend to group accordingly a larger local view into a visual feeling on the whole. In [54], pooling is realized right by the simple practice of down-sampling. A widely adopted way of pooling is *max pooling* [55] namely to take the maximum output value of a local patch of lower-layer units and assign it to the corresponding higher-layer unit.
- *Multiple feature maps*: When talking about parameter sharing and partial parameter sharing above, the author assumes by default that only a single feature map (i.e. a matrix of units) is configured at each layer. This default configuration is adopted by the basic version of the convolutional neural network. A feature map, which has units with the same set of local connection weights but with independent biases, is intended for

learning to extract a kind of shift-invariant feature. Instead of configuring a single feature map at a layer, one may pack multiple feature maps into three-dimensional tensor at the layer, for sake of learning to extract multiple kinds of shift-invariant features.

Even many years before renaissance of deep learning, the convolutional neural network method had found its way in important commercial applications [56], especially character and digit recognition for check reading, optical character recognition (OCR) etc. A number of object recognition techniques based on convolutional neural networks have also been developed ever after.

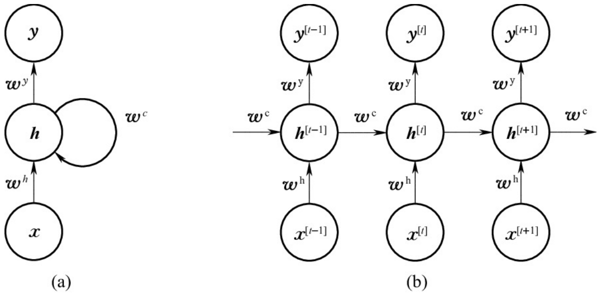
The author has mentioned for several times renaissance of deep learning which has taken place in recent years. If people have to choose a concrete event that marks renaissance of deep learning, the author believes that most people would choose the debut of the AlexNet [57] that won the ImageNet object recognition challenge [58]. The AlexNet has several aspects of contributions to the development of convolutional neural networks: It advocates usage of the *rectified linear unit* or *ReLU* activation function that characterizes non-saturating nonlinearity instead of the *logistic* activation function that characterizes saturating nonlinearity (see Appendix G) and advocates usage of the *dropout* technique, though both the ReLU and the dropout technique had already been available before the debut of the AlexNet. It proposes a local response normalization technique that aims at generalization and a data augmentation technique that aims at reducing over-fitting. Yet the most prominent contributions that the AlexNet has made consist in the following two aspects:

- *GPU implementation*: Deep neural networks have ever since been implemented on *graphic processing units* (GPU) , which demonstrates how hardware progress can revolutionize the landscape of algorithms.
- *Being deep*: The AlexNet initiates the tendency of making artificial neural networks deeper. It consists of five convolutional layers and three full connection (or dense) layers—This configuration looks rather moderate nowadays yet was “amazingly deep” at its birth—It has a huge number of parameters, more specifically, a total number of 60 million parameters .[28] It is this contribution that explicitly coins the term “deep” learning. Following this tendency initiated by the AlexNet, the VGGNet [59] advocates usage of even deeper architectures but only with simple 3×3 convolution kernels. For example, the VGGNet-19 consists of sixteen convolutional layers and three full connection layers, having a total number of 159 million parameters.[29]

As the artificial neural networks become deeper and deeper, the *degradation problem* of training (optimization) becomes more and more severe. In other words, training accuracy suffers from saturation and degrades rapidly as the depth increases. The ResNet [60] demonstrates a methodology of *deep residual learning* to handle the degradation problem.

Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)

Recurrent neural networks [52, 61, 62] are dedicated to sequence learning. In the context of computer vision, sequences can be camera videos, visually tracked objects, etc. Compared with a three-layers (one-hidden layer) artificial neural network, a recurrent neural network has one more “ingredient” namely a self-connection circle from the hidden layer to itself, as illustrated in Fig. 2.39a.



$\boldsymbol{y}$
$\boldsymbol{w}^y$
$\boldsymbol{h}$
$\boldsymbol{w}^c$
$\boldsymbol{w}^h$
$\boldsymbol{x}$
(a)
$\boldsymbol{y}^{[t-1]}$
$\boldsymbol{y}^{[t]}$
$\boldsymbol{y}^{[t+1]}$
$\boldsymbol{w}^y$
$\boldsymbol{w}^c$
$\boldsymbol{h}^{[t-1]}$
$\boldsymbol{h}^{[t]}$
$\boldsymbol{h}^{[t+1]}$
$\boldsymbol{w}^h$
$\boldsymbol{x}^{[t-1]}$
$\boldsymbol{x}^{[t]}$
$\boldsymbol{x}^{[t+1]}$
(b)

Fig. 2.39 Recurrent neural network **a** folded RNN; **b** unfolded RNN

Notations in Fig. 2.39 bear the same meanings as presented in Sect. G.2.2 in Appendix G: The vectors $\boldsymbol{x}$, $\boldsymbol{h}$, and $\boldsymbol{y}$ denote the input layer, the hidden layer, and the output layer respectively. The matrix $\boldsymbol{w}^h$ denotes the weights (including biases) of connections from the input layer to the hidden layer, whereas the matrix $\boldsymbol{w}^y$ denotes the weights (including biases) of connections from the hidden layer to the output layer. One extra notation, i.e. the matrix $\boldsymbol{w}^c$ denotes the weights (including biases) of connections from the hidden layer to itself. The recurrent neural network can be unfolded to highlight its sequential configuration, as illustrated in Fig. 2.39b.

Like (G.15), the recurrent neural network forward mapping can be formalized as

$$\boldsymbol{h}^{[t]I} = \boldsymbol{w}^h \boldsymbol{x}^{[t]A} + \boldsymbol{w}^c \boldsymbol{h}^{[t-1]O} \tag{2.78a}$$

$$\boldsymbol{h}^{[t]O} = 1 \oslash \left(1 + \mathrm{e}^{[-\boldsymbol{h}^{[t]I}]}\right) \tag{2.78b}$$

$$\boldsymbol{y}^{[t]I} = \boldsymbol{w}^y \left(\boldsymbol{h}^{[t]O}\right)^A \tag{2.78c}$$

$$\boldsymbol{y}^{[t]O} = 1 \oslash \left(1 + \mathrm{e}^{[-\boldsymbol{y}^{[t]I}]}\right) \tag{2.78d}$$

Like (G.14), the recurrent neural network back-propagation in t-decrementing order can be formalized as

$$\frac{\partial E}{\partial \boldsymbol{y}^{[t]O}} = \boldsymbol{y}^{[t]O} - \boldsymbol{y}^{[t]} \tag{2.79a}$$

$$\frac{\partial E}{\partial \boldsymbol{y}^{[t]I}} = \frac{\partial E}{\partial \boldsymbol{y}^{[t]O}} \odot \left(1 - \boldsymbol{y}^{[t]O}\right) \odot \boldsymbol{y}^{[t]O} \tag{2.79b}$$

$$\frac{\partial E}{\partial \boldsymbol{h}^{[t]O}} = [\boldsymbol{I} \quad \boldsymbol{0}] \left(\boldsymbol{w}^y\right)^{\mathrm{T}} \frac{\partial E}{\partial \boldsymbol{y}^{[t]I}} + [\boldsymbol{I} \quad \boldsymbol{0}] \left(\boldsymbol{w}^c\right)^{\mathrm{T}} \frac{\partial E}{\partial \boldsymbol{h}^{[t+1]I}} \tag{2.79c}$$

$$\frac{\partial E}{\partial \boldsymbol{h}^{[t]I}} = \frac{\partial E}{\partial \boldsymbol{h}^{[t]O}} \odot \left(1 - \boldsymbol{h}^{[t]O}\right) \odot \boldsymbol{h}^{[t]O} \tag{2.79d}$$

and

$$\frac{\partial E}{\partial \boldsymbol{w}^y} = \sum_t \frac{\partial E}{\partial \boldsymbol{y}^{[t]I}} \left[\left(\boldsymbol{h}^{[t]O}\right)^{\mathrm{T}} \quad 1\right] \tag{2.80a}$$

$$\frac{\partial E}{\partial \boldsymbol{w}^c} = \sum_t \frac{\partial E}{\partial \boldsymbol{h}^{[t+1]I}} \left[\left(\boldsymbol{h}^{[t]O}\right)^{\mathrm{T}} \quad 1 \right] \tag{2.80b}$$

$$\frac{\partial E}{\partial \boldsymbol{w}^h} = \sum_t \frac{\partial E}{\partial \boldsymbol{h}^{[t]I}} \left[\boldsymbol{x}^{[t]\mathrm{T}} \quad 1 \right] \tag{2.80c}$$

It is after iterations of (2.79) in t-decrementing order are completed that (2.80) is executed. The *Hamadard product* $\odot$, the *Hamadard division* $\oslash$, and the *element-wise exponential function* $\mathrm{e}^{[\cdot]}$ are used in (2.78), (2.79), and (2.80).

The standard recurrent neural network architecture tends to suffer from the limited context access problem and the vanishing gradient problem [61]. The **long short-term memory** [63], which takes advantage of memory blocks, is an elegant method to overcome limitations of the standard recurrent neural network architecture.

Some explanations hover over “deepness” of recurrent neural networks. Such deepness may not be obvious in the folded architecture as illustrated in Fig. 2.39a, but can be easily seen in the unfolded architecture as illustrated in Fig. 2.39b. Such deepness is rather a kind of temporal deepness, besides which one can also enable the recurrent neural network to be spatially deep, i.e. having more layers even in the folded architecture.

Autoencoders (AE) and Generative Adversarial Networks (GAN)

Autoencoders [64–66] which include undercomplete autoencoders (i.e. those with code dimension less than the input dimension), regularized autoencoders, sparse autoencoders, denoising autoencoders, contractive autoencoders, and variational autoencoders are another representative family in the domain of deep learning. **Generative adversarial networks** [67] which resort to adversarial learning with the help of discriminator networks are also a representative family for deep learning.

2.8.3 Learn for Feature Extraction

In fact, the idea of incorporating machine learning into feature extraction is not brand new. Since even two decades ago, researchers have been taking advantage of machine learning in extraction of primitive geometric entities such as edges in the image [68]. In fact, the complete versions of both the FAST [27] presented in Sect. 2.7.2 and the BRIEF [28] presented in Sect. 2.7.3 should also have incorporated machine learning to optimize their realization in data-driven way. It is to facilitate readers’ understanding of principal spirits of the FAST and the BRIEF that the machine learning “ingredients” which may distract readers’ attentions are intentionally removed from presentations of the FAST and the BRIEF in Sect. 2.7.2 and Sect. 2.7.3. The author refrains from deviating further into detailed presentation of machine learning based feature extraction. For recommendation, readers may refer to a recent article [69] to know more about machine learning based features.

Data-driven classification based objection detection, which usually relies on machine learning, can somehow be regarded as a kind of feature extraction. By convention, a component of such kind outputs rectangular features with “descriptors”, i.e. labels indicating categories of objects inside them. Typical works include Haar-wavelet-like responses based face detection [48, 49] and histograms of oriented gradients (HOG) based human detection [70]. The renaissance of deep learning has further cultivated prosperity in the area of object detection in the last decade [71, 72]. However, the author refrains from deviating into detailed presentation of classification based objection detection either.

2.9 Note

Extra clarifications and explanations are needed at the end of Chap. 2.

Existing kinds of features and feature extraction methods, even those representative and well-known ones among them, are much more than what have been presented in Chap. 2. On one hand, it is unnecessary, impossible, and even inappropriate to traverse too many of them in this single book. After all, existing features normally differ from each other in how they are represented and how they are extracted, but have no essential difference in assuming the role of concrete geometric entities to serve practical applications. For example, for the image stitching effect demonstrated in Fig. 2.36, the ORB features assume the role of geometric anchors from which the homography (see Chap. 4) between the two image planes can be revealed. Not only ORB features, other proper features such as SIFT features can also assume such geometric role.[30] If one replaces ORB features by other proper features and even by manually selected features, there will be no essential change in how image stitching is performed.

On the other hand, it would be inappropriate either to totally skip presentation of representative feature extraction methods. It is true that one may rely purely on manual operations especially manual selection of key features in images[31] and knowledge presented in following chapters can still hold. However, this will make code demonstrations (which play an important role in facilitating readers' understanding of relevant knowledge) of following chapters too far away from real practice. Still take image stitching as example, it would be ridiculous that one has to manually select some features in captured images when one uses a camera or a smart phone to generate a panorama photo in daily life. Despite its ridiculousness, such practice is at least "theoretically" possible because humans are there. For many other examples which involve full automation without humans, such as autonomous navigation of mobile robots, humans' manual interactions are impossible at all.

Now with a knowledge of representative feature extraction methods that can support a large variety of practical applications and can especially support code demonstrations of following chapters, the author moves on to present various aspects of knowledge that are more related to geometry fundamentals of computer vision than knowledge presented in Chap. 2.

References

1. R.A. Horn, C.R. Johnson. *Topics in Matrix Analysis* (Cambridge University Press, 1991)
2. D. Bradley, G. Roth, Adaptive thresholding using the integral image. J. Graph. Tools **12**(2), 13–21 (2007) [Crossref]
3. J. Canny, A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986) [ADS][Crossref][PubMed]
4. C. Harris, M. Stephens, A combined corner and edge detector, in *Alvey Vision Conference* (1988), pp. 147–151
5. D. Lowe, Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004) [Crossref]
6. D. Forsyth, J. Ponce. *Computer Vision: A Modern Approach* (Pearson Education, 2012)
7. K. Mikolajczyk, *Detection of Local Features Invariant to Affine Transformations* (Institut National Polytechnique de Grenoble, Thèse, 2002)
8. M. Brown, D. Lowe. Invariant features from interest point groups, in *British Machine Vision Conference* (2002)
9. D. Hilbert, *Grundzüge Einer Allgemeinen Theorie der Linearen Integralgleichungen* (Mathematisch-Physikalische Klasse, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, 1904), pp.213–259 [zbMATH]

10.
A. Sedaghat, H. Ebadi, Remote sensing image matching based on adaptive binning sift descriptor. IEEE Trans. Geosc. Remote Sens. **53**(10), 5283–5293 (2015)
[ADS][Crossref]

11.
S. Gupta, K. Thakur, M. Kumar, 2d-human face recognition using sift and surf descriptors of face’s feature regions. The Visual Comput. **37**, 447–456 (2021)
[Crossref]

12.
Z. Long, X. Zhang, M. He, S. Huang, G. Qin, D. Song, Y. Tang, G. Wu, W. Liang, H. Shao, Motor fault diagnosis based on scale invariant image features. IEEE Trans. Ind. Inf. **18**(3), 1605–1617 (2022)
[Crossref]

13.
Y. Fang, J. Liu, J. Li, J. Cheng, J. Hu, D. Yi, X. Xiao, U. Bhatti, Robust zero-watermarking algorithm for medical images based on sift and bandelet-dct. Multimedia Tools Appl. **81**, 16863–16879 (2022)
[Crossref]

14.
Y. Ke, R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors, in *IEEE Conference on Computer Vision and Pattern Recognition* (2004)

15.
J. Morel, GYu. Asift, A new framework for fully affine invariant image comparison. SIAM J. Imaging Sci. **2**(2), 438–469 (2009)

16.
M. Brown, S. Süsstrunk, Multi-spectral sift for scene category recognition, in *IEEE Conference on Computer Vision and Pattern Recognition* (2011), pp. 177–184

17.
J. Ma, J. Zhao, J. Jiang, H. Zhou, X. Guo, Locality preserving matching. Int. J. Comput. Vis. **127**, 512–531 (2019)
[MathSciNet][Crossref]

18.
C. Laurgeau. *Le siècle de la Voiture Intelligente* (Presses des MINES, 2009)

19.
A. Eskandarian (ed.), *Handbook of Intelligent Vehicles* (Springer-Verlag, London, 2012)

20.
B. Siciliano, O. Khatib. *Springer Handbook of Robotics* (Springer, 2016)

21.
R. Deriche. Recursively implementing the gaussian and its derivatives. *INRIA Research Report*, RR-1893 (1993)

22.
R. Horaud, O. Monga. *Vision par Ordinateur: Outils Fondamentaux* (Editions Hermès, 1995)

23.
H. Bay, A. Ess, T. Tuytelaars, L. Gool, Speeded-up robust features (surf). Comput. Vis. Image Underst. **110**(3), 346–359 (2008)
[Crossref]

24.
Y. Meyer, R. Ryan. *Wavelets: Algorithms and Applications* (SIAM, 1993)

25.
S. Mallat, A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**(7), 674–693 (1989)
[ADS][Crossref][zbMATH]

26.
E. Rosten, T. Drummond, Fusing points and lines for high performance tracking, in *IEEE International Conference on Computer Vision* (2005), pp. 1508–1515

27.
E. Rosten, R. Porter, T. Drummond, Faster and better: a machine learning approach to corner detection. IEEE Trans. Pattern Anal. Mach. Intell. **32**(1), 105–119 (2010)
[ADS][Crossref][PubMed]

28.
M. Calonder, V. Lepetit, C. Strecha, P. Fua. Brief: binary robust independent elementary features, in *European Conference on Computer Vision*, (2010), pp. 778–792

29.
R. Hamming, Error detecting and error correcting codes. Bell Syst. Tech. J. **29**(2), 147–160 (1950)
[ADS][MathSciNet][Crossref][zbMATH]

30.
E. Rublee, V. Rabaud, K. Konolige, G. Bradski. Orb: an efficient alternative to sift or surf, in *IEEE International Conference on Computer Vision*, (2011), pp. 2564–2571

31.
R. Mur-Artal, J. Montiel, J. Tardós, Orb-slam: a versatile and accurate monocular slam system. IEEE Trans. Robot. **31**(5), 1147–1163 (2015)
[Crossref]

32.
R. Mur-Artal, J. Tardós, Orb-slam2: an open-source slam system for monocular, stereo, and rgb-d cameras. IEEE Trans. Robot. **33**(5), 1255–1262 (2017)
[Crossref]

33. C. Campos, R. Elvira, J. Rodríguez, J. Montiel, J. Tardós, Orb-slam3: an accurate open-source library for visual, visual-inertial, and multimap slam. IEEE Trans. Robot. **37**(6), 1874–1890 (2021)
[Crossref]

34. P. Rosin, Measuring corner properties. Comput. Vis. Image Understand. **73**(2), 291–307 (1999)
[Crossref]

35. Y. Le Cun, Y. Bengio, G. Hinton, Deep learning. Nature **521**, 436–444 (2015)
[ADS][Crossref]

36. I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning* (MIT Press, 2016)

37. S. Theodoridis, K. Koutroumbas. *Pattern Recognition* (Elsevier, 2006)

38. I. Jolliffe. *Principal Component Analysis* (Springer Science & Business Media, 1986)

39. A. Hyvärinen, E. Oja, Independent component analysis: algorithms and applications. Neural Netw. **13**(4–5), 411–430 (2000)
[Crossref][PubMed]

40. P. Paatero, U. Tapper, Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values. Environmetrics **5**(2), 111–126 (1994)
[Crossref]

41. V. Vapnik. *The Nature of Statistical Learning Theory* (Springer-Verlag, 1995)

42. M. Best. *Quadratic Programming with Computer Programs* (CRC Press, 2017)

43. G. Dantzig. *Linear Programming and Extensions*. *US Air Force, Tech Report*, R-366-PR (1963)

44. L. Valiant, A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)
[Crossref][zbMATH]

45. R. Schapire, The strength of weak learnability. Mach. Learning **5**, 197–227 (1990)
[Crossref]

46. Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
[MathSciNet][Crossref][zbMATH]

47. H. Li, F. Nashashibi, M. Yang, Split covariance intersection filter: theory and its application to vehicle localization. IEEE Trans. Intell. Transp. Syst. **14**(4), 1860–1871 (2013)
[Crossref]

48. P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 511–518 (2001)

49. P. Viola, M. Jones, Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
[Crossref]

50. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386–408 (1958)
[Crossref][PubMed]

51. M. Minsky, S. Papert, Perceptrons (MIT Press, 1969)

52. D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
[ADS][Crossref]

53. D. Rumelhart, G. Hinton, R. Williams, Learning internal representations by error propagation. Parallel Distributed Processing **1**, 318–362 (1986)

54. Y. Le Cun. Generalization and network design strategies. *University of Toronto, Tech Report*, CRG-TR-89-4, 1989

55. Y. Zhou, R. Chellappa, Computation of optical flow using a neural network. IEEE Int. Conf. Neural Netw. **2**, 71–78 (1988)
[Crossref]

56. Y. Le Cun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
[ADS][Crossref]

57. A. Krizhevsky, I. Sutskever, G. Hinton, *Imagenet classification with deep convolutional neural networks* (Adv. Neural Inf. Proc, Syst, 2012)

58. J. Deng, W. Dong, R. Socher, L. Li, K. Li, F. Li, Imagenet: a large-scale hierarchical image database, in *IEEE Computer Vision and Pattern Recognition* (2009), pp. 248–255

59. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *International Conference on Learning Representations* (2015)

60. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778

61. A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks* (Springer-Verlag, Berlin Heidelberg, 2012)
[Crossref][zbMATH]

62. I. Sutskever. *Training Recurrent Neural Networks* (Ph.D. Thesis, University of Toronto, 2013)

63. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
[Crossref][PubMed]

64. Y. Le Cun, F. Fogelman-Soulié, Modèles connexionnistes de l'apprentissage. Intellectica **1**(2–3), 114–143 (1987)

65. G. Hinton R. Zemel, Autoencoders, minimum description length and helmholtz free energy, in *Advances in Neural Information Processing Systems*, (1993), pp. 3–10

66. D. Kingma, M. Welling, Auto-encoding variational bayes, in *International Conference on Learning Representations* (2014)

67. I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in *International Conference on Learning Representations* (2015)

68. S. Konishi, A. Yuille, J. Coughlan, S. Zhu, Statistical edge detection: learning and evaluating edge cues. IEEE Trans. Pattern Anal. Mach. Intell. **25**(1), 57–74 (2003)
[ADS][Crossref]

69. J. Ma, X. Jiang, A. Fan, J. Jiang, J. Yan, Image matching from handcrafted to deep features: a survey. Int. J. Comput. Vis. **129**, 23–79 (2021)
[MathSciNet][Crossref]

70. N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in *IEEE Conference on Computer Vision and Pattern Recognition* (2005), pp. 886–893

71. Z. Zhao, P. Zheng, S. Xu, X. Wu, Object detection with deep learning: a review. IEEE Trans. Neural Netw. Learning Syst. **30**(11), 3212–3232 (2019)
[Crossref]

72. L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, M. Pietikäinen, Deep learning for generic object detection: a survey. Int. J. Comput. Vis. **128**, 261–318 (2020)
[Crossref][zbMATH]

Footnotes

[1] $\lfloor\rfloor$ denotes the floor function. For example, $\lfloor \pi \rfloor = 3$, $\lfloor -e \rfloor = -3$.

[2] $\bar{u}$ denotes the closest integer to u, namely the "bar" on head denotes the round function. For example, $\bar{\pi} = 3$, $\bar{e} = 3$.

[3] Matrix vectorization:

$$\mathrm{vec}([\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \cdots \quad \boldsymbol{b}_n]) \equiv \begin{bmatrix} cb_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$ where $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, ..., $\boldsymbol{b}_n$ are vectors.

[4] Kronecker product: $\boldsymbol{A} \otimes \boldsymbol{B} \equiv \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B} \end{bmatrix}$. Then $\mathrm{vec}(\boldsymbol{ABC}) = (\boldsymbol{C}^{\mathrm{T}} \otimes \boldsymbol{A})\mathrm{vec}(\boldsymbol{B})$.

[5] In the code, explicit m-script implementation of the *Kronecker product* function $\otimes$ is given, simply to facilitate readers' understanding of Kronecker product. In fact, the Matlab built-in function *kron* can be used directly to perform the same function.

[6] This is why "Eyes are windows of mind", because visual data do convey plenty of information.

[7] "Tribute to Yan-An" is composed by the author of this book and is published on *China Daily*.

[8] If a point has a negative value, then it is compared with neighbour points also with negative values and their absolute values are compared.

[9] Intensities of the gradient images are augmented by 20 times for better visualization.

[10] In literature, the Laplacian operator may sometimes be denoted also as ∇^2, yet the author prefers the notation Δ here, because the author would like to reserve the notation ∇^2 for the Hessian matrix.

[11] Write the variable differential transform in matrix form as $\begin{bmatrix} \mathrm{d}u' \\ \mathrm{d}v' \end{bmatrix} = \begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix} \begin{bmatrix} \mathrm{d}r \\ \mathrm{d}\theta \end{bmatrix}$. The determinant of the transform matrix is $\begin{vmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} = r$, this is why the bivariate differential expression $\mathrm{d}u'\mathrm{d}v'$ becomes $r\mathrm{d}r\mathrm{d}\theta$ after the variable transform.

[12] Usually to validate certain extremum of a function via infinitesimal analysis, terms only up to two orders are considered and definiteness of the Hessian matrix is checked. However, as analysed below, the Hessian matrix of the Laplacian ΔI_{DF} is degenerate at the extremum (u_0, v_0), which makes it impossible to validate the extremum simply according to the Hessian matrix. So terms up to more orders are considered.

[13] Researches have demonstrated that extrema of the scale-invariant Laplacian produce not only salient but also stable features [7].

[14] Strictly speaking, it is when σ^2 is treated holistically as a variable that the parametric Gaussian filter satisfies the heat equation.

[15] Given a specific σ, the scale-invariant Laplacian of its associated Gaussian filtered image is referred to as its associated **scale-invariant Laplacian layer**.

[16] This is a conventional practice for scale-space feature extraction.

[17] The German term "eigen" (in "eigenwerte") was introduced by *Hilbert* in [9], which means "own", "proprietary" (sense extended from its French origin "propre"), or "characteristic". This new "English" term has been adopted as mathematics convention ever since.

[18] Take a positive definite $\nabla^2 D$ as example. Let e_1 and e_2 denote the larger and smaller eigenvalues of $\nabla^2 D$ respectively, note the index

$$\frac{\mathrm{tr}(\nabla^2 D)^2}{\det(\nabla^2 D)} = \frac{(e_1+e_2)^2}{e_1 e_2} = \frac{(e_1/e_2+1)^2}{e_1/e_2},$$

which increases monotonically with e_1/e_2 when $e_1/e_2 > 1$. So the condition that e_1/e_2 is beyond (or within) certain threshold is equivalent to the condition that tr $(\nabla^2 D)^2/$ det $(\nabla^2 D)$ is beyond (or within) certain corresponding threshold.

[19] The σ parameter associated with the SIFT feature point, i.e. $\tau^k \sigma_0$ is the scale indicator of the SIFT feature point, yet for clear visualization, the line segment length is actually set to six times the σ parameter $\tau^k \sigma_0$ instead of $\tau^k \sigma_0$ directly.

[20] Not only SIFT feature points, for these or those kinds of feature points, establishment of correspondence among multiple sets of feature points is commonly called **registration** or **matching**.

[21] *A posteriori*, one can intuitively observe corresponding SIFT feature points among different images. However, a priori, the computer vision system does not know corresponding SIFT feature points among different images. For SIFT feature points in one image, the computer vision system must have certain way to match them with SIFT feature points in another image respectively and regard matched SIFT feature points as corresponding SIFT feature points, be the matched SIFT feature points actually corresponding or not.

[22] In fact, for the case that the scale differs by a factor of 2^j $(j = 1, 2, \cdots)$, i.e. an exponential of two among images, the scaling factor τ^{k-1} in (2.60) can be saved as well. However, except for artificial synthesis, such coincidence is rarely encountered in practical applications.

[23] Different from that generated randomly, a refined or even “optimal” set of sampling patterns for BRIEF description may be obtained in data-driven way via machine learning.

[24] The Matlab built-in function *bitxor* does bit-wise XOR operation.

[25] Details of image stitching are omitted here but will be introduced in this book. It is worth noting that in the stitched image, slight color inconsistency along the boundary of the image viewing towards the left side is intentionally kept to let readers see the “influencing” areas of both original images in the stitched image. Such slight color inconsistency itself can be easily handled in practice.

[26] The support vector machine method is intended by default for bi-classification.

[27] The weak learners are sometimes just a little better than random guesses. Assumption of independence among weak learners is unnecessary (and usually impossible) for boosting to take effect, yet it is true that weak learner should have certain extent of independence partially instead of having total correlation to be able to indeed complement each other somehow. For conceptualization of such partial independence and partial correlation by analogy, readers may borrow relevant concepts about state estimation from *control theory* [47].

[28] The first convolutional layer consists of two tensors of size $55 \times 55 \times 48$ (i.e. 48 feature maps of size 55×55) accommodated by two GPUs respectively, with a convolution kernel of size $11 \times 11 \times 3$; so the layer has $(11 \times 11 \times 3 + 55 \times 55) \times 48 \times 2 = 325248$ parameters. The second convolutional layer consists of two tensors of size $27 \times 27 \times 128$ on the two GPUs respectively, with a convolution kernel of size $5 \times 5 \times 48$ (input from the previous layer only on one GPU); so the layer has $(5 \times 5 \times 48 + 27 \times 27) * 128 * 2 = 493824$ parameters. The third convolutional layer consists of two tensors of size $13 \times 13 \times 192$, with a convolution kernel of size $3 \times 3 \times 256$ (input from the previous layer on both GPUs); so the layer has $(3 \times 3 \times (128 + 128) + 13 \times 13) \times 192 \times 2 = 949632$ parameters. The fourth

convolutional layer consists of two tensors of size $13 \times 13 \times 192$, with a convolution kernel of size $3 \times 3 \times 192$ (input from the previous layer only on one GPU); so the layer has $(3 \times 3 \times 192 + 13 \times 13) \times 192 \times 2 = 728448$ parameters. The fifth convolutional layer consists of two tensors of size $13 \times 13 \times 128$, with a convolution kernel of size $3 \times 3 \times 192$ (input from the previous layer only on one GPU); so the layer has $(3 \times 3 \times 192 + 13 \times 13) \times 128 \times 2 = 485632$ parameters. After max pooling, the fifth convolutional layer has two output tensors of size $6 \times 6 \times 128$. The first full connection layer has two arrays of 2048 neurons distributed on the two GPUs respectively; so the layer has $6 \times 6 \times (128 + 128) \times (2048 + 2048) + (2048 + 2048) = 37752832$ parameters. The second full connection layer has two arrays of 2048 neurons; so the layer has $4096 \times 4096 + 4096 = 16781312$ parameters. The third full connection layer has 1000 neurons; so the layer has $4096 \times 1000 + 1000 = 4097000$ parameters. Therefore, the AlexNet has a total number of $325248 + 493824 + 949632 + 728448 + 485632 + 37752832 + 16781312 + 4097000 = 61613928$ parameters ≈ 60 million parameters.

[29] The sixteen convolutional layers are grouped into five blocks: two layers (tensors) of size $224 \times 224 \times 64$ in the first block, two layers (tensors) of size $112 \times 112 \times 128$ in the second block, four layers (tensors) of size $56 \times 56 \times 256$ in the third block, four layers (tensors) of size $28 \times 28 \times 512$ in the fourth block, and four layers (tensors) of size $14 \times 14 \times 512$ in the fifth block. Max pooling is concatenated at the end of each block, shrinking the spatial scale by a factor of 2. The three full connection layers are the same to those in the AlexNet. So the VGGNet-19 has a total number of $(3 \times 3 \times 3 + 224^2) \times 64 + (3 \times 3 \times 64 + 224^2) \times 64 + (3 \times 3 \times 64 + 112^2) \times 128 + (3 \times 3 \times 128 + 112^2) * 128 + (3 \times 3 \times 12$ parameters ≈ 159 million parameters—The original “Table 2” in [59] gives only the numbers of connection weights, not considering the numbers of feature map biases.

[30] In fact, SIFT features tend to outperform ORB features for image stitching.

[31] For many practical applications, incorporation of manual operations is indeed a not bad kind of choice, compared with fully automatic computer vision solutions.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_3

3. Computational Geometry

Hao Li[1]
(1) Shanghai Jiao Tong University, Shanghai, China

3.1 Distance
3.1.1 Euclidean Distance
3.1.2 Minimum-Distance-Path Perspective
3.1.3 Point-to-Line Distance
3.1.4 Point-to-Plane Distance
3.2 Generalized Distances and Measures
3.2.1 L_p-Norm Distance
3.2.2 City-Block Distance
3.2.3 Mahalanobis Distance
3.2.4 Distance, Area, Volume, and Fractal Measures
3.3 Graph
3.3.1 Topological Abstraction
3.3.2 Planar Graphs and Euler Formula
3.3.3 Planar Straight-Line Graphs
3.3.4 Point-Location Problem
3.4 Convex Geometry
3.4.1 Convex Polygon
3.4.2 Point-Inclusion Problem
3.5 Closest Point Problems
3.5.1 Find the Closest Pair Among Points
3.5.2 Voronoi Diagram
References

In practical applications of computer vision, one will inevitably encounter geometric entities and needs to know how to analyse them theoretically, especially needs to know how to represent and compute their geometric properties. Besides, to envisage and analyse geometric entities theoretically in abstract way is one thing, whereas to put what one can envisage and analyse into practical representations and computations is another thing. The second thing is well related to but is still considerably different from the first thing.

For simple example, one can fairly envisage a line and a separate point, and fairly states like “traverse all points on the line and obtain the one closest to the separate point”.[1] Despite its logical fairness, one can never realize the statement literally, even with the help of the most powerful supercomputer available in the world. The impossibility is simply due to the fact that one will never be able to really traverse all the infinite points on the line and check them one by one.

This simple example tells that a gap between what one can envisage and analyse for geometric entities and what one can realize (compute) for them does exist. **Computational geometry** [2, 3] aims right at bridging such a gap, namely to effectively realize the essence of what one can envisage and analyse, if essentially realizable.

3.1 Distance

In practical applications of computer vision, one may need to deal with certain *distances* related to concerned geometric entities. Some distances that one needs to handle may take the literal physical sense of the concept distance according to daily-life understanding, whereas other distances under consideration may take certain abstract sense for sake of characterizing "closeness" (similarity) or "apartness" (distinction) among concerned geometric entities, be the geometric entities themselves concrete or abstract.

3.1.1 Euclidean Distance

The Euclidean distance is the conventionally-adopted distance in daily-life practices. Given two one-dimensional points

$$\boldsymbol{P}_1 = [x_1], \quad \boldsymbol{P}_2 = [x_2],$$

their Euclidean distance is

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = |x_1 - x_2|.$$

Given two two-dimensional points

$$\boldsymbol{P}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix},$$

their Euclidean distance is

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Given two three-dimensional points

$$\boldsymbol{P}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix},$$

their Euclidean distance is

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

Given two generic points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ in a *n*-dimensional vector space

$$\boldsymbol{P}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{bmatrix},$$

the **Euclidean distance** between the two points is defined as

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{(x_{1,1} - x_{2,1})^2 + (x_{1,2} - x_{2,2})^2 + \cdots + (x_{1,n} - x_{2,n})^2} \tag{3.1}$$

which can also be formalized compactly as

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2} = \sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)}.$$

The definition of the Euclidean distance in (3.1) is identical to the definition of the L_2-norm of the difference vector $\boldsymbol{P}_1 - \boldsymbol{P}_2$. This is why the L_2-norm notation $\| \cdot \|_2$ is adopted above to denote the Euclidean distance.

Besides the condition of *positive definiteness*, the condition of *triangular inequality* namely

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\| \leqslant \|\boldsymbol{P}_1 - \boldsymbol{P}_3\| + \|\boldsymbol{P}_3 - \boldsymbol{P}_2\| \tag{3.2}$$

or formalized equivalently as

$$\|\boldsymbol{a} + \boldsymbol{b}\| \leqslant \|\boldsymbol{a}\| + \|\boldsymbol{b}\|$$

is the other necessary condition that the distance metric must satisfy. The Euclidean distance satisfies the triangular inequality

$$\begin{aligned}
&\sqrt{(a_1 + b_1)^2 + \cdots + (a_n + b_n)^2} \leqslant \sqrt{a_1^2 + \cdots + a_n^2} + \sqrt{b_1^2 + \cdots + b_n^2} \\
\Leftrightarrow\quad & (a_1 + b_1)^2 + \cdots + (a_n + b_n)^2 \leqslant \left(\sqrt{a_1^2 + \cdots + a_n^2} + \sqrt{b_1^2 + \cdots + b_n^2}\right)^2 \\
\Leftrightarrow\quad & a_1 b_1 + \cdots + a_n b_n \leqslant \sqrt{a_1^2 + \cdots + a_n^2}\sqrt{b_1^2 + \cdots + b_n^2} \quad \text{(Cauchy inequality)} \\
\Leftrightarrow\quad & (a_1 b_1 + \cdots + a_n b_n)^2 \leqslant (a_1^2 + \cdots + a_n^2)(b_1^2 + \cdots + b_n^2) \\
\Leftrightarrow\quad & 2\sum_{1\leqslant i<j\leqslant n} a_i b_i a_j b_j \leqslant \sum_{1\leqslant i<j\leqslant n} (a_i^2 b_j^2 + a_j^2 b_i^2) \Longleftrightarrow 0 \leqslant \sum_{1\leqslant i<j\leqslant n} (a_i b_j - a_j b_i)^2
\end{aligned}$$

which holds apparently.

3.1.2 Minimum-Distance-Path Perspective

Readers may wonder why the triangular inequality is indispensable for definition of the distance. This can be explained from the minimum-distance-path perspective. Before the author continues the explanation, one had better first reflect on the essence of the so-called “distance”.

The author has been talking about the distance abstractly in mathematical terms by so far, but in reality, what is people’s intuitive conception of the distance? Imagine when people say that the distance from Wuhan to Shenyang is larger than the distance from Wuhan to Beijing. The author believes that people hardly think in the following way, i.e. first recalling the distance from Wuhan to Shenyang is roughly 1600 km, then recalling the distance from Wuhan to Beijing is roughly 800 km, and finally concluding the former is larger than the latter. Instead, people would have a direct intuitive conception that it takes more time to travel (by high-speed train for example) from Wuhan to Shenyang than from Wuhan to Beijing. In other words, people’s intuitive conception of the distance between two geographic points is related to the time needed to travel from one point to the other—Such intuitive conception of the distance “measured” by time can also be well reflected in daily-life expressions such as a distance of “one hour by car”, of “ten minutes by walk” etc. [2]

Besides, when one talks about the time needed to travel from Wuhan to Beijing by high-speed train, nobody would find such expression ambiguous, though in reality there are many ways of travelling from Wuhan to Beijing by high-speed train (say travelling from Wuhan to Beijing directly, travelling from Wuhan to Shanghai first and then to Beijing, or travelling from Wuhan consecutively to Xi’an, to Taiyuan, and finally to Beijing) and the different ways have different travelling time. Why does not the expression “the time needed to travel from Wuhan to Beijing by high-speed train” pose any ambiguity to daily-life communication? The reason is that people mean and understand the “time needed” in the expression by default as the *least* or *minimum* time needed, without confusing ourselves with meaningless “time needed” for arbitrary ways of travelling. Such minimum time needed is obviously related to the minimum one among the distances of all paths that connect Wuhan to Beijing, namely related to the distance of the minimum-distance-path.

Therefore, the distance can be conceived from the **minimum-distance-path** perspective: the distance between two points is the distance of the minimum-distance-path that connects the two points. In fact, this conception can be further generalized to specifying not only the distance between two points but also that between a point and a point set, that between two point sets, and can also be further generalized to specifying distances not only in the Euclidean space but also in a more generalized space such as the *Riemannian space* .[3] Latter, when describing the point-to-line distance and the point-to-plane distance, the author will adopt the minimum-distance-path perspective.

Then return to the indispensable role that the triangular inequality plays in definition of the distance. In a word, the triangular inequality guarantees the minimum-distance requirement. Given two generic points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ in a generic space, as illustrated in Fig. 3.1, the distance is defined as $\|\boldsymbol{P}_2 - \boldsymbol{P}_1\|$. Now consider an arbitrary path that connects from $\boldsymbol{P}_1$ to $\boldsymbol{P}_2$, illustrated by the path that passes consecutively from $\boldsymbol{P}_1$ to $\boldsymbol{P}_3$, $\boldsymbol{P}_4$, ..., $\boldsymbol{P}_{n-1}$, $\boldsymbol{P}_n$, and finally to $\boldsymbol{P}_2$. Thanks to the triangular inequality, the distance of this arbitrary path satisfies

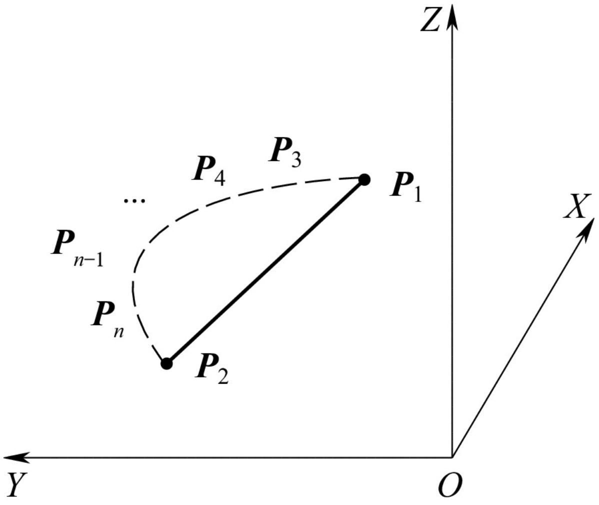


Fig. 3.1 Minimum-distance-path perspective

$$
\begin{aligned}
\text{path distance} &= \|\boldsymbol{P}_3-\boldsymbol{P}_1\|+\|\boldsymbol{P}_4-\boldsymbol{P}_3\|+\cdots+\|\boldsymbol{P}_n-\boldsymbol{P}_{n-1}\|+\|\boldsymbol{P}_2-\boldsymbol{P}_n\| \\
&\geqslant \|\boldsymbol{P}_4-\boldsymbol{P}_1\|+\cdots+\|\boldsymbol{P}_n-\boldsymbol{P}_{n-1}\|+\|\boldsymbol{P}_2-\boldsymbol{P}_n\| \\
&\geqslant \quad \cdots \\
&\geqslant \|\boldsymbol{P}_n-\boldsymbol{P}_1\|+\|\boldsymbol{P}_2-\boldsymbol{P}_n\| \\
&\geqslant \|\boldsymbol{P}_2-\boldsymbol{P}_1\|.
\end{aligned}
$$

In other words, the distance of an arbitrary path that connects the two points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ is at least $\|\boldsymbol{P}_2-\boldsymbol{P}_1\|$, or the distance $\|\boldsymbol{P}_2-\boldsymbol{P}_1\|$ is indeed the distance of the minimum-distance-path, consistent with the minimum-distance requirement.

On the other hand, imagine the triangular inequality is violated for certain distance metric, then it would be meaningless to specify the distance between two points, because one may potentially find a shorter-distance path that connects the two points and that shorter-distance plays a role more like “distance” than the so-called distance specified. Finally, one’s analysis based on such distance metric will be lost in a total chaos.

3.1.3 Point-to-Line Distance

Since the Euclidean distance is the conventionally-adopted distance in daily-life practices, without explicit mentioning, the distance metric refers to the Euclidean distance by default. The point-to-point distance is defined in (3.1). How to compute the point-to-line distance?

Given three two-dimensional points

$$
\boldsymbol{P}_1=\begin{bmatrix}x_1\\ y_1\end{bmatrix},\quad \boldsymbol{P}_2=\begin{bmatrix}x_2\\ y_2\end{bmatrix},\quad \boldsymbol{P}=\begin{bmatrix}x\\ y\end{bmatrix}.
$$

Consider the line spanned by the points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ and the distance of the point $\boldsymbol{P}$ to the line. Any point on the line spanned by $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ can be parametrized as

$$
\boldsymbol{P}(t)=t\boldsymbol{P}_1+(1-t)\boldsymbol{P}_2=\begin{bmatrix}(x_1-x_2)t+x_2\\ (y_1-y_2)t+y_2\end{bmatrix}.
$$

From the minimum-distance-path perspective, the distance between $\boldsymbol{P}$ and the line is

$$
d=\min\|\boldsymbol{P}(t)-\boldsymbol{P}\|_2=\min\sqrt{((x_1-x_2)t+x_2-x)^2+((y_1-y_2)t+y_2-y)^2}. \tag{3.3}
$$

Solve

$$
\begin{aligned}
t &= \operatorname{argmin}_t\ ((x_1-x_2)t+x_2-x)^2+((y_1-y_2)t+y_2-y)^2 \\
&= \operatorname{argmin}_t\ ((x_1-x_2)^2+(y_1-y_2)^2)t^2+2((x_1-x_2)(x_2-x)+(y_1-y_2)(y_2-y))t \\
&\qquad\qquad +(x_2-x)^2+(y_2-y)^2 \\
&= \frac{(x_1-x_2)(x-x_2)+(y_1-y_2)(y-y_2)}{(x_1-x_2)^2+(y_1-y_2)^2}=\frac{(\boldsymbol{P}_1-\boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}-\boldsymbol{P}_2)}{(\boldsymbol{P}_1-\boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1-\boldsymbol{P}_2)}.
\end{aligned} \tag{3.4}
$$

Substitute (3.4) into (3.3) to compute the point-to-line distance in the two-dimensional space.

Given three three-dimensional points

$$\boldsymbol{P}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Consider the line spanned by the three-dimensional points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ and the distance of the three-dimensional point $\boldsymbol{P}$ to the line. Any point on the line spanned by $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ can be parametrized as

$$\boldsymbol{P}(t) = t\boldsymbol{P}_1 + (1-t)\boldsymbol{P}_2 = \begin{bmatrix} (x_1 - x_2)t + x_2 \\ (y_1 - y_2)t + y_2 \\ (z_1 - z_2)t + z_2 \end{bmatrix}.$$

From the minimum-distance-path perspective, the distance between $\boldsymbol{P}$ and the line is

$$\begin{aligned} d &= \min \|\boldsymbol{P}(t) - \boldsymbol{P}\|_2 \\ &= \min \sqrt{((x_1-x_2)t+x_2-x)^2+((y_1-y_2)t+y_2-y)^2+((z_1-z_2)t+z_2-z)^2}. \end{aligned} \tag{3.5}$$

Solve

$$\begin{aligned} t = \; & \operatorname{argmin}_t \; ((x_1-x_2)t+x_2-x)^2+((y_1-y_2)t+y_2-y)^2+((z_1-z_2)t+z_2-z)^2 \\ = \; & \operatorname{argmin}_t \; ((x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2)t^2 \\ & +2((x_1-x_2)(x_2-x)+(y_1-y_2)(y_2-y)+(z_1-z_2)(z_2-z))t \\ & +(x_2-x)^2+(y_2-y)^2+(z_2-z)^2 \\ = \; & \frac{(x_1-x_2)(x-x_2)+(y_1-y_2)(y-y_2)+(z_1-z_2)(z-z_2)}{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2} \\ = \; & \frac{(\boldsymbol{P}_1-\boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}-\boldsymbol{P}_2)}{(\boldsymbol{P}_1-\boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1-\boldsymbol{P}_2)}. \end{aligned} \tag{3.6}$$

Substitute (3.6) into (3.5) to compute the point-to-line distance in the three-dimensional space.

Given three generic points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, and $\boldsymbol{P}$ in a n-dimensional vector space

$$\boldsymbol{P}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Consider the line spanned by the n-dimensional points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ and the distance of the n-dimensional point $\boldsymbol{P}$ to the line. Any point on the line spanned by $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ can be parametrized as

$$\boldsymbol{P}(t) = t\boldsymbol{P}_1 + (1-t)\boldsymbol{P}_2 = (\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2.$$

From the minimum-distance-path perspective, the distance between $\boldsymbol{P}$ and the line is

$$\begin{aligned} d &= \min \|\boldsymbol{P}(t) - \boldsymbol{P}\|_2 = \min \|(\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2 - \boldsymbol{P}\|_2 \\ &= \min \sqrt{((\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2 - \boldsymbol{P})^{\mathrm{T}}((\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2 - \boldsymbol{P})}. \end{aligned} \tag{3.7}$$

Solve

$$\begin{aligned} t &= \operatorname{argmin}_t \; ((\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2 - \boldsymbol{P})^{\mathrm{T}}((\boldsymbol{P}_1 - \boldsymbol{P}_2)t + \boldsymbol{P}_2 - \boldsymbol{P}) \\ &= \operatorname{argmin}_t \; (\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)t^2 + 2(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_2 - \boldsymbol{P})t + (\boldsymbol{P}_2 - \boldsymbol{P})^{\mathrm{T}}(\boldsymbol{P}_2 - \boldsymbol{P}) \\ &= \frac{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P} - \boldsymbol{P}_2)}{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)}. \end{aligned} \tag{3.8}$$

So the projection of the point $\boldsymbol{P}$ on the line spanned by the points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ is

$$\boldsymbol{P}_{\text{proj}} = \frac{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P} - \boldsymbol{P}_2)}{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)}(\boldsymbol{P}_1 - \boldsymbol{P}_2) + \boldsymbol{P}_2. \tag{3.9}$$

Substitute (3.8) into (3.7) to compute the point-to-line distance in the *n*-dimensional space

$$d = \sqrt{\frac{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)(\boldsymbol{P} - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P} - \boldsymbol{P}_2) - ((\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P} - \boldsymbol{P}_2))^2}{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)}}. \tag{3.10}$$

To judge whether the projection of the point $\boldsymbol{P}$ on the line is inside the line segment spanned by the points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, one can check whether t obtained via (3.8) is inside the interval [0, 1], i.e. whether $0 \leqslant t \leqslant 1$.

Moreover, to compute the distance between $\boldsymbol{P}$ and the line segment, first compute t via (3.8), then truncate t into the interval [0, 1] as

$$t = \begin{cases} 0, & t < 0 \\ t, & 0 \leqslant t \leqslant 1 \\ 1, & t > 1 \end{cases}$$

and finally substitute the truncated t into (3.7) to compute the distance between the point and the line segment.

3.1.4 Point-to-Plane Distance

Besides the point-to-point distance, the point-to-line distance, how to compute the point-to-plane distance? Given $n + 1$ generic points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$, and $\boldsymbol{P}$ in an n-dimensional vector space

$$\boldsymbol{P}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{bmatrix}, \quad \cdots \quad, \quad \boldsymbol{P}_n = \begin{bmatrix} x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,n} \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

View the *n* points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$ as *n* vectors and suppose they are linearly independent, namely suppose the compound matrix formed by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$

$$\boldsymbol{C} \equiv [\boldsymbol{P}_1 \quad \boldsymbol{P}_2 \quad \cdots \quad \boldsymbol{P}_n] = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,n} & x_{2,n} & \cdots & x_{n,n} \end{bmatrix} \tag{3.11}$$

is invertible.

Consider the $(n-1)$-dimensional *hyperplane* or for short the *plane* spanned by the *n* points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$. Since the compound matrix $\boldsymbol{C}$ specified in (3.11) is assumed invertible, such spanned plane is not a degenerate but a well-formed $(n-1)$-dimensional hyperplane.

Any point on the plane spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$ can be parametrized as

$$\begin{aligned} \boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{C}\ \boldsymbol{t} \equiv [\boldsymbol{P}_1 \quad \boldsymbol{P}_2 \quad \cdots \quad \boldsymbol{P}_n]\boldsymbol{t} \equiv & \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,n} & x_{2,n} & \cdots & x_{n,n} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \\ \text{s.t.} \qquad \boldsymbol{b}^{\mathrm{T}}\boldsymbol{t} = t_1 + t_2 + \cdots + t_n = 1, & \end{aligned} \tag{3.12}$$

where

$$\begin{aligned} \boldsymbol{t} &\equiv [t_1 \quad t_2 \quad \cdots \quad t_n]^{\mathrm{T}}, \\ \boldsymbol{b} &\equiv [1 \quad 1 \quad \cdots \quad 1]^{\mathrm{T}}. \end{aligned}$$

From the minimum-distance-path perspective, the distance between $\boldsymbol{P}$ and the plane spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$ is

$$\begin{aligned} d = \quad & \min \|\boldsymbol{P}(\boldsymbol{t}) - \boldsymbol{P}\|_2 = \min \|\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P}\|_2 = \min \sqrt{(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P})^{\mathrm{T}}(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P})} \\ & \text{s.t.} \qquad \boldsymbol{b}^{\mathrm{T}}\boldsymbol{t} = 1. \end{aligned} \tag{3.13}$$

Solve the constrained optimization problem

$$\begin{aligned} \boldsymbol{t} = \quad & \operatorname{argmin}_{\boldsymbol{t}} \ \tfrac{1}{2}(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P})^{\mathrm{T}}(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P}) \\ & \text{s.t.} \qquad \boldsymbol{b}^{\mathrm{T}}\boldsymbol{t} = 1. \end{aligned} \tag{3.14}$$

Construct a *Lagrange multiplier* [4]

$$L(\boldsymbol{t}, \lambda) = \tfrac{1}{2}(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P})^{\mathrm{T}}(\boldsymbol{C}\ \boldsymbol{t} - \boldsymbol{P}) - \lambda(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{t} - 1),$$

which gives the (first) optimality condition

$$\tfrac{\partial L}{\partial \boldsymbol{t}} \quad = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{t} - \boldsymbol{C}^{\mathrm{T}}\boldsymbol{P} - \lambda\boldsymbol{b} = 0, \tag{3.15a}$$

$$\tfrac{\partial L}{\partial \lambda} \quad = \boldsymbol{b}^{\mathrm{T}}\boldsymbol{t} - 1 = 0. \tag{3.15b}$$

The first equation of (3.15) gives

$$\boldsymbol{t} = (\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C})^{-1}(\boldsymbol{C}^{\mathrm{T}}\boldsymbol{P} + \lambda\boldsymbol{b}) = \boldsymbol{C}^{-1}(\boldsymbol{P} + \lambda\boldsymbol{C}^{-\mathrm{T}}\boldsymbol{b}) = \boldsymbol{C}^{-1}(\boldsymbol{P} + \lambda\bar{\boldsymbol{b}})$$

where

$$\bar{\boldsymbol{b}} \equiv \boldsymbol{C}^{-\mathrm{T}}\boldsymbol{b}$$

and substitute it into the second equation of (3.15)

$$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{C}^{-1}(\boldsymbol{P} + \lambda\bar{\boldsymbol{b}}) = 1 \Longleftrightarrow \bar{\boldsymbol{b}}^{\mathrm{T}}(\boldsymbol{P} + \lambda\bar{\boldsymbol{b}}) = 1 \Longleftrightarrow \lambda = \frac{1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P}}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}$$

So the solution $\boldsymbol{t}$ of (3.15) is

$$\boldsymbol{t} = \boldsymbol{C}^{-1}\left[\boldsymbol{P} + \frac{(1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P})}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}\bar{\boldsymbol{b}}\right]. \tag{3.16}$$

The projection of $\boldsymbol{P}$ on the plane spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$ is

$$\boldsymbol{P}_{\text{proj}} = \boldsymbol{C}\ \boldsymbol{C}^{-1}\left[\boldsymbol{P} + \frac{(1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P})}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}\bar{\boldsymbol{b}}\right] = \boldsymbol{P} + \frac{(1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P})}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}\bar{\boldsymbol{b}}. \tag{3.17}$$

The distance between $\boldsymbol{P}$ and the plane spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$ is

$$d = \sqrt{(\boldsymbol{P}_{\text{proj}} - \boldsymbol{P})^{\mathrm{T}}(\boldsymbol{P}_{\text{proj}} - \boldsymbol{P})}. \tag{3.18}$$

Matlab code for computing the parameter vector $\boldsymbol{t}$ via (3.16), the projection via (3.17), and the distance via (3.18) is given as follows.

PointToPlaneDistance.m

```
%% Compute point-to-plane geometric properties
% C: Compound matrix of P1, P2, ..., Pn that span the plane (
    hyperplane)
% P: Target point
% Pr: Projection of P on the plane spanned by P1, P2, ..., Pn
% tPr:Coefficients of Pr expressed in terms of P1, P2, ..., Pn
% dP: Distance between P and the plane spanned by P1, P2, ...,
    Pn
function [dP, tPr, Pr] = PointToPlaneDistance(C, P)
    n = length(P); invC = inv(C); b = transpose(invC)*ones(n,1);
    Pr = P+((1-transpose(b)*P)/(transpose(b)*b))*b;
    tPr = invC*Pr;
    dP = sqrt(transpose(Pr-P)*(Pr-P));
end
```

Point-Plane Topological Relationship

To judge whether a given point $\boldsymbol{P}$ is located on the plane spanned by n given points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$, resort to (3.17) and check if

$$\begin{aligned}\boldsymbol{P}_{\text{proj}} = \boldsymbol{P}\ &\Longleftrightarrow \boldsymbol{P} + \frac{(1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P})}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}\bar{\boldsymbol{b}} = \boldsymbol{P} \Longleftrightarrow \frac{(1-\bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P})}{\bar{\boldsymbol{b}}^{\mathrm{T}}\bar{\boldsymbol{b}}}\bar{\boldsymbol{b}} = \boldsymbol{0}\\ &\Longleftrightarrow 1 - \bar{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{P} = 0 \Longleftrightarrow \boldsymbol{b}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{P} = 1.\end{aligned} \tag{3.19}$$

In other words, compute the coefficients of $\boldsymbol{P}$ expressed linearly in terms of $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ... , $\boldsymbol{P}_n$

$$\boldsymbol{t}_{\boldsymbol{P}} = \boldsymbol{C}^{-1}\boldsymbol{P}$$

and then check whether the summation of the elements of $\boldsymbol{t}_{\boldsymbol{P}}$ is equal to 1.

Point-Triangle Topological Relationship

In practical applications, one may need to judge whether a given point is inside a given triangle or not. More specifically, consider the three-dimensional vector space and four points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$, and $\boldsymbol{P}$, one needs to judge whether $\boldsymbol{P}$ is inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ or not—Suppose the three points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ do not belong to the degenerate case.

First, check whether $\boldsymbol{P}$ is located on the plane spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ according to (3.19). Compute the coefficients of $\boldsymbol{P}$ expressed linearly in terms of $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$

$$\boldsymbol{t}_{\boldsymbol{P}} \equiv \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_1 & \boldsymbol{P}_2 & \boldsymbol{P}_3 \end{bmatrix}^{-1} \boldsymbol{P} \tag{3.20}$$

and check if

$$t_1 + t_2 + t_3 = 1. \tag{3.21}$$

If the summation of the elements of $\boldsymbol{t}_{\boldsymbol{P}}$ is not equal to 1, then $\boldsymbol{P}$ is definitely not inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$. On the other hand, if the summation is equal to 1, then further check if the condition

$$t_1 \geqslant 0, \quad t_2 \geqslant 0, \quad t_3 \geqslant 0 \tag{3.22}$$

is satisfied or not. If the condition (3.22) is satisfied, $\boldsymbol{P}$ is inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$; otherwise, not inside.

If the three points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ belong to the degenerate case,[4] then there are two potential ways to analyse the point-triangle topological relationship. The first way is to vary all relevant points by the same arbitrary amount of translation and then analyse the translated points in the same way as presented above. Validity of the first way relies on the fact that a common translation will never change the point-triangle topological relationship as well as the fact that a translation will make the three points escape from the degenerate case only if the translation is not along a direction parallel to the plane spanned by the three points.

The second way is to compute the coefficients of the augmented $\boldsymbol{P}$ expressed linearly in terms of the augmented $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$

$$\boldsymbol{t}_{\boldsymbol{P}} \equiv \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_1 & \boldsymbol{P}_2 & \boldsymbol{P}_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{P} \\ 1 \end{bmatrix} \tag{3.23}$$

and then check whether the solved coefficients satisfy both the normalization condition (3.21) and the non-negative condition (3.22). It is worth noting that in practical applications, the non-negative condition (3.22) is rarely used alone but tends to be accompanied by the normalization condition (3.21). Since the normalization condition (3.21) is usually adopted by default, the non-negative condition (3.22) is actually equivalent to the following condition

$$0 \leqslant t_1 \leqslant 1, \quad 0 \leqslant t_2 \leqslant 1, \quad 0 \leqslant t_3 \leqslant 1.$$

In fact, be the three points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ belonging to the degenerate case or not, the coefficients t_1, t_2, t_3 can always be computed by solving (3.23). Then the normalization condition (3.21) and the non-negative condition (3.22) are checked to analyse the topological relationship between the point $\boldsymbol{P}$ and the triangle spanned by the three points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$. **Planar Point-Triangle Topological Relationship**

The method can also be adapted to judge on a two-dimensional plane whether a given point is inside a given triangle. Given four points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$, and $\boldsymbol{P}$ on the two-dimensional plane

$$\boldsymbol{P}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \quad \boldsymbol{P}_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} x \\ y \end{bmatrix}.$$

and suppose $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ are not colinear.

To judge whether $\boldsymbol{P}$ is inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$, augment the four points by adding 1 to their tails

$$\boldsymbol{P}_1^{\mathrm{A}} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_2^{\mathrm{A}} = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_3^{\mathrm{A}} = \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}^{\mathrm{A}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Compute the coefficients t_1, t_2, t_3 of $\boldsymbol{P}^{\mathrm{A}}$ expressed linearly in terms of $\boldsymbol{P}_1^{\mathrm{A}}$, $\boldsymbol{P}_2^{\mathrm{A}}$, $\boldsymbol{P}_3^{\mathrm{A}}$

$$\begin{aligned} \begin{bmatrix} \boldsymbol{P}_1^{\mathrm{A}} & \boldsymbol{P}_2^{\mathrm{A}} & \boldsymbol{P}_3^{\mathrm{A}} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} &= \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ \Longrightarrow \quad \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{P}_1^{\mathrm{A}} & \boldsymbol{P}_2^{\mathrm{A}} & \boldsymbol{P}_3^{\mathrm{A}} \end{bmatrix}^{-1} \boldsymbol{P}^{\mathrm{A}} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \end{aligned}$$

According to the third row of the matrices in above equality, the summation of t_1, t_2, t_3 is guaranteed to be equal to 1. In other words, the condition (3.21) is satisfied naturally. Just check the condition (3.22) to judge whether $\boldsymbol{P}$ is inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$.

For example, consider three points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$

$$\boldsymbol{P}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}, \quad \boldsymbol{P}_3 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

which span a triangle, as illustrated in Fig. 3.2. Judge whether two extra points $\boldsymbol{P}_4$ and $\boldsymbol{P}_5$

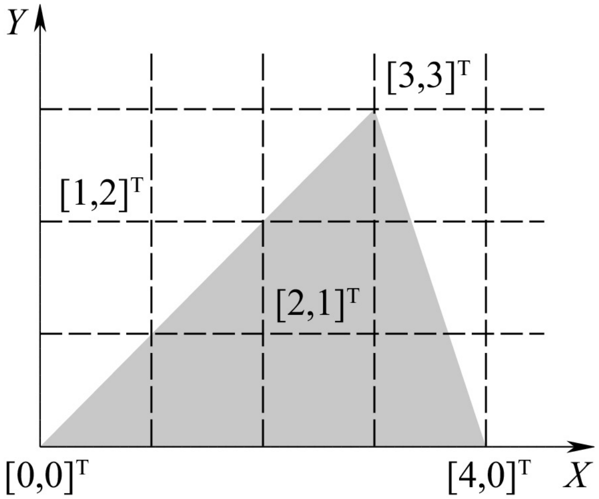


Fig. 3.2 Judgement of whether points are inside a triangle

$$\boldsymbol{P}_4 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \boldsymbol{P}_5 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

are inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$ or not.

First, augment these points to

$$\boldsymbol{P}_1^{\mathrm{A}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_2^{\mathrm{A}} = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_3^{\mathrm{A}} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_4^{\mathrm{A}} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{P}_5^{\mathrm{A}} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}.$$

Compute

$$\begin{aligned} \boldsymbol{t}_{\boldsymbol{P}_4} &= \left[\boldsymbol{P}_1^{\mathrm{A}} \quad \boldsymbol{P}_2^{\mathrm{A}} \quad \boldsymbol{P}_3^{\mathrm{A}}\right]^{-1} \boldsymbol{P}_4^{\mathrm{A}} = \begin{bmatrix} 0 & 4 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 7/12 \\ -1/4 \\ 2/3 \end{bmatrix}, \\ \boldsymbol{t}_{\boldsymbol{P}_5} &= \left[\boldsymbol{P}_1^{\mathrm{A}} \quad \boldsymbol{P}_2^{\mathrm{A}} \quad \boldsymbol{P}_3^{\mathrm{A}}\right]^{-1} \boldsymbol{P}_5^{\mathrm{A}} = \begin{bmatrix} 0 & 4 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5/12 \\ 1/4 \\ 1/3 \end{bmatrix}. \end{aligned}$$

The second element of $\boldsymbol{t}_{\boldsymbol{P}_4}$ violates the condition (3.22), so $\boldsymbol{P}_4$ is not inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$. In contrast, $\boldsymbol{t}_{\boldsymbol{P}_5}$ satisfies the condition (3.22), so $\boldsymbol{P}_5$ is inside the triangle spanned by $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, $\boldsymbol{P}_3$.

3.2 Generalized Distances and Measures

Generalized distances can be defined via generalization of the Euclidean distance from various perspectives. Given two generic points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ in a n-dimensional vector space

$$\boldsymbol{P}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{bmatrix}, \quad \boldsymbol{P}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{bmatrix}.$$

As defined in (3.1), the Euclidean distance between the two points is formalized as

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{(x_{1,1} - x_{2,1})^2 + (x_{1,2} - x_{2,2})^2 + \cdots + (x_{1,n} - x_{2,n})^2}$$

or compactly as

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{\sum_{i=1}^{n} (x_{1,i} - x_{2,i})^2} = \sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}}(\boldsymbol{P}_1 - \boldsymbol{P}_2)}.$$

If the formalism of the Euclidean distance is viewed as a special case of the formalism

$$\begin{aligned} \|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 &= \sqrt[p]{|x_{1,1} - x_{2,1}|^p + |x_{1,2} - x_{2,2}|^p + \cdots + |x_{1,n} - x_{2,n}|^p} \\ &= (|x_{1,1} - x_{2,1}|^p + |x_{1,2} - x_{2,2}|^p + \cdots + |x_{1,n} - x_{2,n}|^p)^{\frac{1}{p}}, \end{aligned}$$

where the superscript, i.e. the exponential (or power) number p of each term in the summation is $p = 2$, then a way of generalizing the Euclidean distance is available. If the formalism of the Euclidean distance is viewed as a special case of the formalism

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_2 = \sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}} \boldsymbol{M}^{-1} (\boldsymbol{P}_1 - \boldsymbol{P}_2)},$$

where the positive definite matrix $\boldsymbol{M}$ is the identity matrix, i.e. $\boldsymbol{M} = \boldsymbol{I}$, then another way of generalizing the Euclidean distance is also available.

3.2.1 L_p-Norm Distance

Generalize the definition of the Euclidean distance by replacing the superscript “2” with a generic superscript “p” ($p \geqslant 1$) to form the **L_p-norm distance**

$$\|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_p = (|x_{1,1} - x_{2,1}|^p + |x_{1,2} - x_{2,2}|^p + \cdots + |x_{1,n} - x_{2,n}|^p)^{\frac{1}{p}}. \tag{3.24}$$

The L_2-norm distance $\|\cdot\|_2$ is right the Euclidean distance.

Any L_p-norm distance ($p \geqslant 1$) satisfies the triangular inequality

$$(|a_1 + b_1|^p + \cdots + |a_n + b_n|^p)^{\frac{1}{p}} \leqslant (|a_1|^p + \cdots + |a_n|^p)^{\frac{1}{p}} + (|b_1|^p + \cdots + |b_n|^p)^{\frac{1}{p}}. \tag{3.25}$$

When $p = 1$, the triangular inequality (3.25) holds apparently. When $p > 1$, the triangular inequality (3.25) is just the *Minkowski inequality* [5], which is proved as

$$\begin{aligned}
& \left(\sum_{i=1}^{n}|a_i+b_i|^p\right)^{\frac{1}{p}} \leqslant \left(\sum_{i=1}^{n}|a_i|^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n}|b_i|^p\right)^{\frac{1}{p}} \\
\Longleftrightarrow\quad & \sum_{i=1}^{n}|a_i+b_i|^p \leqslant \\
& \left[\left(\sum_{i=1}^{n}|a_i|^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n}|b_i|^p\right)^{\frac{1}{p}}\right]\left(\sum_{i=1}^{n}|a_i+b_i|^p\right)^{1-\frac{1}{p}} \quad \left(\frac{1}{q} \equiv 1-\frac{1}{p}\right) \\
\Longleftarrow\quad & \sum_{i=1}^{n}(|a_i||a_i+b_i|^{p-1}+|b_i||a_i+b_i|^{p-1}) \leqslant \\
& \left[\left(\sum_{i=1}^{n}|a_i|^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{n}|b_i|^p\right)^{\frac{1}{p}}\right]\left(\sum_{i=1}^{n}|a_i+b_i|^p\right)^{\frac{1}{q}} \\
\Longleftarrow\quad & \sum_{i=1}^{n}|c_i||a_i+b_i|^{p-1} \leqslant \left(\sum_{i=1}^{n}|c_i|^p\right)^{\frac{1}{p}}\left(\sum_{i=1}^{n}|a_i+b_i|^p\right)^{\frac{1}{q}} \quad (c \in \{a,b\}) \\
\Longleftrightarrow\quad & \sum_{i=1}^{n}|c_i||a_i+b_i|^{p-1} \leqslant \left(\sum_{i=1}^{n}|c_i|^p\right)^{\frac{1}{p}}\left(\sum_{i=1}^{n}(|a_i+b_i|^{p-1})^q\right)^{\frac{1}{q}}
\end{aligned}$$

which holds according to the *Hölder inequality* [5]

$$\sum_{i=1}^{n}\alpha_i\beta_i \leqslant \left(\sum_{i=1}^{n}\alpha_i^p\right)^{\frac{1}{p}}\left(\sum_{i=1}^{n}\beta_i^q\right)^{\frac{1}{q}} \quad \left(\forall\alpha_i,\beta_i \geqslant 0, \frac{1}{p}+\frac{1}{q}=1, p>1\right). \tag{3.26}$$

Note that given $x \geqslant 0$, $y \geqslant 0$, and $\frac{1}{p}+\frac{1}{q}=1$ with $p>1$, then

$$x^{\frac{1}{p}}y^{\frac{1}{q}} \leqslant \frac{x}{p}+\frac{y}{q}, \tag{3.27}$$

which can be verified by checking the extremum and the convexity of the univariate function

$$f(x) = \frac{x}{p}+\frac{y}{q}-x^{\frac{1}{p}}y^{\frac{1}{q}}$$

or the bivariate function

$$f(x,y) = \frac{x}{p}+\frac{y}{q}-x^{\frac{1}{p}}y^{\frac{1}{q}}.$$

Then the Hölder inequality (3.26) is equivalent to

$$
\begin{aligned}
& \sum_{i=1}^{n}\left(\frac{\alpha_i^p}{\sum_{i=1}^{n}\alpha_i^p}\right)^{\frac{1}{p}}\left(\frac{\beta_i^q}{\sum_{i=1}^{n}\beta_i^q}\right)^{\frac{1}{q}} \leqslant 1 \\
\Longleftarrow \quad & \sum_{i=1}^{n}\left(\frac{\alpha_i^p}{\sum_{i=1}^{n}\alpha_i^p}\right)^{\frac{1}{p}}\left(\frac{\beta_i^q}{\sum_{i=1}^{n}\beta_i^q}\right)^{\frac{1}{q}} \leqslant \sum_{i=1}^{n}\left(\frac{1}{p}\frac{\alpha_i^p}{\sum_{i=1}^{n}\alpha_i^p}+\frac{1}{q}\frac{\beta_i^q}{\sum_{i=1}^{n}\beta_i^q}\right) \\
= \quad & \tfrac{1}{p}+\tfrac{1}{q}=1,
\end{aligned}
$$

where (3.27) is applied for each term in the summation on the left side.

Besides the Euclidean distance, i.e. the L_2-norm distance, other kinds of L_p-norm distances may also be adopted for practical applications. For example, the obstacle distance map to be presented in Sect. 3.2.2 usually adopts the L_1-norm distance instead of the Euclidean distance. The *principal component analysis* may rely on various L_p-norm optimization [6, 7]. The L_∞-norm distance constraint also finds its way in adversarial attacks oriented machine learning for computer vision [8, 9].

3.2.2 City-Block Distance

The city-block distance between two two-dimensional points

$$
\boldsymbol{P}_1=\begin{bmatrix}x_1\\y_1\end{bmatrix},\quad \boldsymbol{P}_2=\begin{bmatrix}x_2\\y_2\end{bmatrix},
$$

is right their L_1-norm distance on the two-dimensional plane

$$
\|\boldsymbol{P}_1-\boldsymbol{P}_2\|_1=|x_1-x_2|+|y_1-y_2|. \tag{3.28}
$$

The term itself is a figurative expression of the two-dimensional L_1-norm distance, as illustrated in Fig. 3.3.

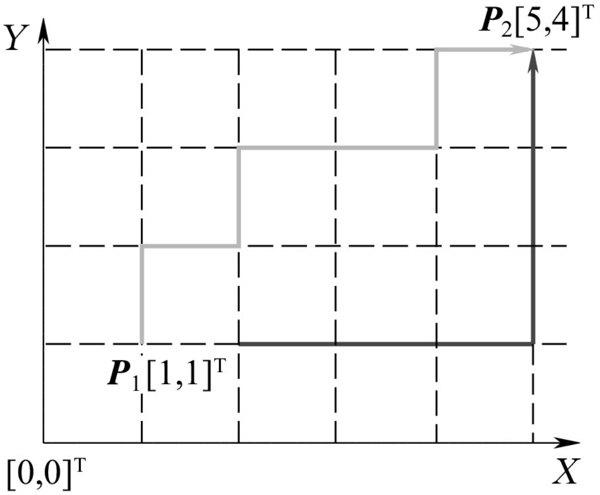

$\boldsymbol{P}_2[5,4]^{\mathrm{T}}$
Y
$\boldsymbol{P}_1[1,1]^{\mathrm{T}}$
$[0,0]^{\mathrm{T}}$
X

Fig. 3.3 City-block distance

For example, given two points $\boldsymbol{P}_1=[1\quad 1]^{\mathrm{T}}$ and $\boldsymbol{P}_2=[5\quad 4]^{\mathrm{T}}$ in a map of city-blocks. If one moves from $\boldsymbol{P}_1$ to $\boldsymbol{P}_2$, she or he must pass at least four blocks horizontally and at least three blocks vertically. Besides, there are $C_{3+4}^{3}=35$ paths by which one passes exactly four blocks horizontally and three blocks vertically. Therefore, the shortest paths from $\boldsymbol{P}_1$ to $\boldsymbol{P}_2$ have the same city-block distance

$$\|\boldsymbol{P}_2 - \boldsymbol{P}_1\|_1 = |5-1| + |4-1| = 4 + 3 = 7.$$

Readers can easily see figurativeness of the term "city-block" distance.[5]

The city-block distance is especially useful for fast establishment of an *obstacle distance map* that describes the distance of each free-space point to the closest obstacle in two-dimensional discrete space. More specifically, given a two-dimensional environment, where obstacles and free space are marked by the black and white parts respectively, as illustrated in Fig. 3.4. Suppose the two-dimensional environment is discretized and hence the obstacles and free space can be represented together by an image called *obstacle map*.

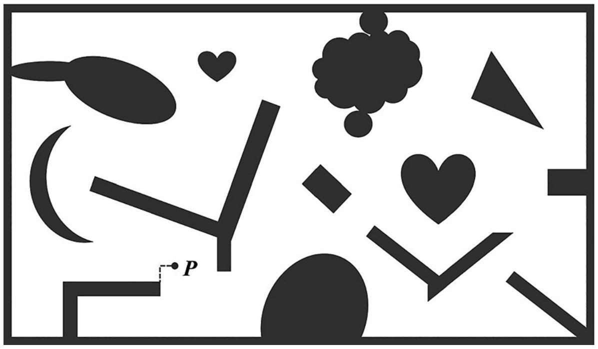


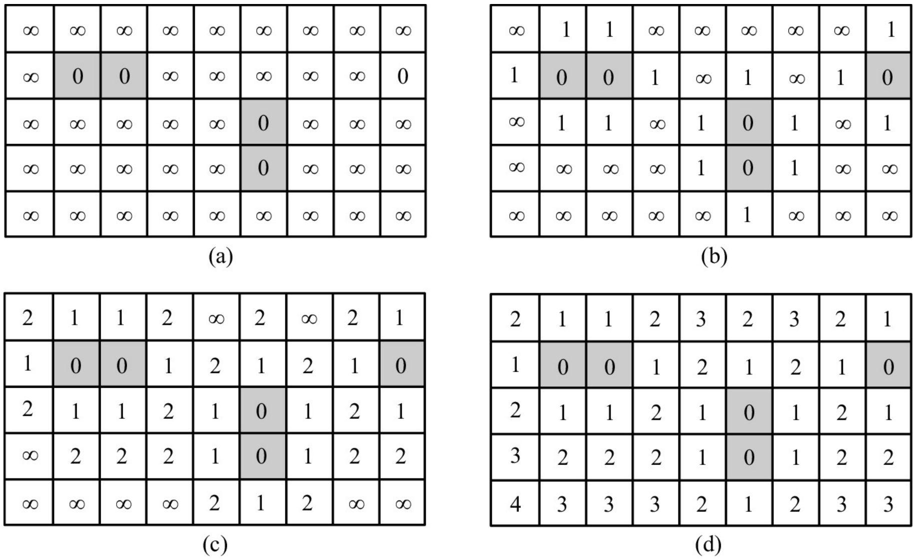
Fig. 3.4 Obstacle map

For a generic point $\boldsymbol{P}$ in free space, its closest (city-block) distance to obstacles is illustrated by the dash line in Fig. 3.4. The task of establishing the obstacle distance map for a given obstacle map is as follows: create a new image of the same size as the obstacle map, and call this new image the obstacle distance map. Then for each free-space point (pixel), i.e. non-obstacle point (pixel), compute its closest distance to obstacles and store the distance value in the corresponding pixel location of the obstacle distance map.

Obviously, pixel locations corresponding to obstacles are easily set to 0. The essence of the task is how to efficiently compute distances associated with free-space points. If the Euclidean distance is adopted as the distance metric for the obstacle distance map, the task would be rather difficult. Fortunately however, in many practical applications, one only needs to have certain distance metric roughly characterizing proximity between free-space points and obstacles, whereas whether such proximity is characterized in people's habitual way (i.e. via the Euclidean distance) is not so cared. For example, for safety of robot navigation, one requires the robot to follow a path (i.e. a sequence of free-space points) of high distance values in the obstacle distance map so that the robot can be far away enough from any obstacle during navigation. Any reasonable distance metric, not necessarily the Euclidean distance, tends to satisfy such requirement for robot navigation.

Therefore, one can adopt the city-block distance rather than the Euclidean distance as the distance metric for the obstacle distance map, and can take advantage of speciality of the city-block distance, namely that the city-block distance between two points is simply a linear combination (summation) of their horizontal and vertical coordinates differences, to facilitate establishment of the obstacle distance map.

To establish an obstacle distance map that adopts the city-block distance, a natural idea is to first extract boundary points of obstacles, use them to initialize a queue (a first-in-first-out structure), and then apply the *breadth-first search* technique together with the queue to update the obstacle distance map gradually from free-space points near obstacles to those far away from obstacles. The process of establishing the obstacle distance map via BFS is demonstrated in Fig. 3.5, where the sub-figure(a) illustrates initialization of the obstacle distance map, the sub-figures(b) and (c) illustrate intermediate obstacle distance maps obtained during the BFS, and the sub-figure(d) illustrates the finally updated obstacle distance map after BFS.



∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	∞	∞	∞	∞	∞	0
∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞

(a)

∞	1	1	∞	∞	∞	∞	∞	1
1	0	0	1	∞	1	∞	1	0
∞	1	1	∞	1	0	1	∞	1
∞	∞	∞	∞	1	0	1	∞	∞
∞	∞	∞	∞	∞	1	∞	∞	∞

(b)

2	1	1	2	∞	2	∞	2	1
1	0	0	1	2	1	2	1	0
2	1	1	2	1	0	1	2	1
∞	2	2	2	1	0	1	2	2
∞	∞	∞	∞	2	1	2	∞	∞

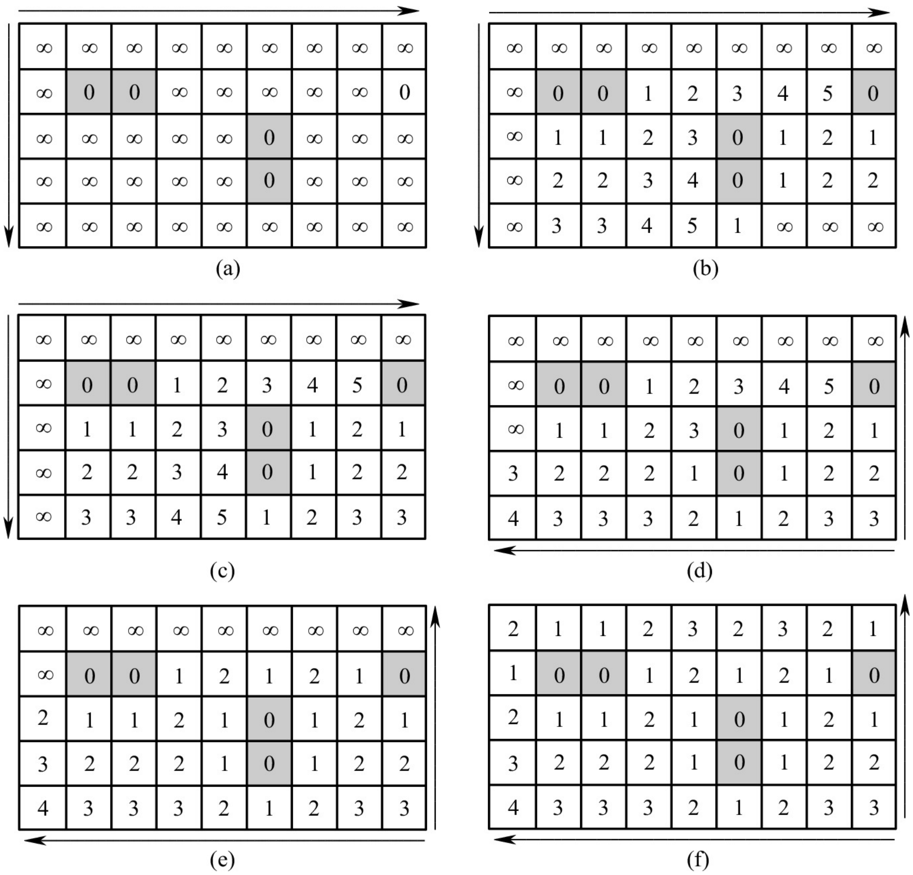
(c)

2	1	1	2	3	2	3	2	1
1	0	0	1	2	1	2	1	0
2	1	1	2	1	0	1	2	1
3	2	2	2	1	0	1	2	2
4	3	3	3	2	1	2	3	3

(d)

Fig. 3.5 Obstacle distance map established via BFS **a** initialization; **b** one round of BFS update; **c** two rounds of BFS update; **d** finally updated obstacle distance map after BFS

On the other hand, a more elegant and efficient method to establish an obstacle distance map that adopts the city-block distance involves only two simple sequential scans of the obstacle map [10]: one from the top-left to the bottom-right, and the other from the bottom-right to the top-left.



∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	∞	∞	∞	∞	∞	0
∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞

(a)

∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	1	2	3	4	5	0
∞	1	1	2	3	0	1	2	1
∞	2	2	3	4	0	1	2	2
∞	3	3	4	5	1	∞	∞	∞

(b)

∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	1	2	3	4	5	0
∞	1	1	2	3	0	1	2	1
∞	2	2	3	4	0	1	2	2
∞	3	3	4	5	1	2	3	3

(c)

∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	1	2	3	4	5	0
∞	1	1	2	3	0	1	2	1
3	2	2	2	1	0	1	2	2
4	3	3	3	2	1	2	3	3

(d)

∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	1	2	1	2	1	0
2	1	1	2	1	0	1	2	1
3	2	2	2	1	0	1	2	2
4	3	3	3	2	1	2	3	3

(e)

2	1	1	2	3	2	3	2	1
1	0	0	1	2	1	2	1	0
2	1	1	2	1	0	1	2	1
3	2	2	2	1	0	1	2	2
4	3	3	3	2	1	2	3	3

(f)

Fig. 3.6 Obstacle distance map established via double sequential scans **a** initialization; **b** intermediate obstacle distance map during the first sequential scan; (c) obstacle distance map after the first sequential scan; **d**, **e** intermediate obstacle distance map during the second sequential scan; **f** finally updated obstacle distance map after double sequential scans

The obstacle distance map is initialized with obstacle and free-space pixels set to 0 and ∞ respectively, as illustrated in Fig. 3.6a. After initialization, the first sequential scan is performed from the top-left to the bottom-right and the obstacle distance map is updated accordingly. More specifically, start from the left-most pixel of the first row, and move right to update pixels of the first row sequentially, until the right-most pixel of the first row is updated. Then move down to scan the next row (i.e. the second row), and update pixels of the second row also sequentially from the left-most one to the right-most one. Move down further to scan and update the next row (i.e. the third row) also in the left-to-right way. Repeat scan and update of rows one by one from the top to the bottom, until the right-most pixel of the last row is updated. During the first sequential scan, each pixel in the obstacle distance map is updated by the minimum one among three distances that are one plus the distance of the pixel above, one plus the distance of the left pixel, and the distance of current pixel, namely

$$I(u,v) := \min\{1 + I(u,v-1), 1 + I(u-1,v), I(u,v)\}, \tag{3.29}$$

.

Once the first sequential scan is finished, the second sequential scan is performed from the bottom-right to the top-left and the obstacle distance map is updated accordingly. More specifically, start from the right-most pixel of the last row, and move left to update pixels of the last row sequentially, until the left-most pixel of the last row is updated. Then move up to scan the previous row (i.e. the second last row), and update pixels of the second last row also sequentially from the right-most one to the left-most one. Move up further to scan and update the previous row (i.e. the third last row) also in the right-to-left way. Repeat scan and update of rows one by one from the bottom to the top, until the left-most pixel of the first row is updated. During the second sequential scan, each pixel in the obstacle distance map is updated by the minimum one among three distances that are one plus the distance of the pixel below, one plus the distance of the right pixel, and the distance of current pixel, namely

$$I(u,v) := \min\left\{1 + I(u,v+1), 1 + I(u+1,v), I(u,v)\right\}. \tag{3.30}$$

Intermediate obstacle distance maps obtained during the second sequential scan are illustrated in Fig. 3.6(d) (e), and the obstacle distance map obtained after the second sequential scan, namely the finally updated obstacle distance map, is illustrated in Fig. 3.6(f).

Matlab code for establishing the obstacle distance map via double sequential scans based on (3.29) and (3.30) respectively is given as follows.

ObstacleDistanceMap.m

```
%% Establish the obstacle distance map
% om: Obstacle map (0 for obstacles and 255 for free space)
% odm: Obstacle distance map that adopts the city-block distance
function odm = ObstacleDistanceMap(om)
    [H,W] = size(om);
    k = find(om<128); om(:,:) = H+W; om(k) = 0;
    odm = H+W-zeros(H+2,W+2); odm(2:H+1,2:W+1) = om; %
    Initialization
    % First sequential scan: from top-left to bottom-right
    for r=2:H+1
        for c=2:W+1
            odm(r,c) = min([1+odm(r-1,c),1+odm(r,c-1),odm(r,c)])
    ;
        end
    end
    % Second sequential scan: from bottom-right to top-left
    for r=H+1:-1:2
        for c=W+1:-1:2
            odm(r,c) = min([1+odm(r+1,c),1+odm(r,c+1),odm(r,c)])
    ;
        end
    end
    odm = odm(2:H+1,2:W+1);
end
```

The function **ObstacleDistanceMap.m** is applied to the obstacle map illustrated in Fig. 3.4. The obtained obstacle distance map that adopts the city-block distance is demonstrated in Fig. 3.7, where pixel intensities are proportional to their distances to obstacles: the brighter the pixel is, the further the corresponding point is from obstacles; the darker the pixel is, the closer the corresponding point is to obstacles.

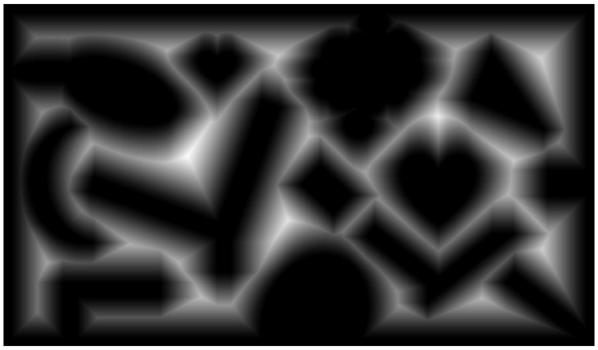


Fig. 3.7 Obstacle distance map

3.2.3 Mahalanobis Distance

Given two generic points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ in a vector space, the **Mahalanobis distance** [11] between the two points is defined as

$$d_{\boldsymbol{M}}(\boldsymbol{P}_1, \boldsymbol{P}_2) \equiv \|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_{\boldsymbol{M}} \equiv \sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}} \boldsymbol{M}^{-1} (\boldsymbol{P}_1 - \boldsymbol{P}_2)}, \tag{3.31}$$

where $\boldsymbol{M}$ denotes certain positive definite matrix.[6]

The Mahalanobis distance is indeed a distance metric. Its positive definiteness is obvious according to its definition (3.31):

$$\sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}} \boldsymbol{M}^{-1} (\boldsymbol{P}_1 - \boldsymbol{P}_2)} \geqslant 0$$

and

$$\sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}} \boldsymbol{M}^{-1} (\boldsymbol{P}_1 - \boldsymbol{P}_2)} = 0 \Longleftrightarrow \boldsymbol{P}_1 - \boldsymbol{P}_2 = 0, \text{ i. e. } \boldsymbol{P}_1 = \boldsymbol{P}_2,$$

because the matrix $\boldsymbol{M}$ is positive definite (so is the matrix inverse $\boldsymbol{M}^{-1}$). Validity of the *triangular inequality* concerning the Mahalanobis distance is proved below.

Proof Since $\boldsymbol{M}^{-1}$ is positive definite (and hence symmetric by default), it can be decomposed into a product of two mutually-transposed matrices as [12]

$$\boldsymbol{M}^{-1} = \boldsymbol{S}^{\mathrm{T}} \boldsymbol{S}.$$

So the formalism of (3.31) can be transformed to

$$d_{\boldsymbol{M}}(\boldsymbol{P}_1, \boldsymbol{P}_2) \equiv \|\boldsymbol{P}_1 - \boldsymbol{P}_2\|_{\boldsymbol{M}} \equiv \sqrt{(\boldsymbol{P}_1 - \boldsymbol{P}_2)^{\mathrm{T}} \boldsymbol{S}^{\mathrm{T}} \boldsymbol{S} (\boldsymbol{P}_1 - \boldsymbol{P}_2)} = \|\boldsymbol{S}(\boldsymbol{P}_1 - \boldsymbol{P}_2)\|_2, \tag{3.32}$$

where $\|\cdot\|_2$ denotes the L_2-norm, i.e. the *Euclidean distance*.

Consider three generic points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, and $\boldsymbol{P}_3$, then

$$\begin{aligned} d_{\boldsymbol{M}}(\boldsymbol{P}_1, \boldsymbol{P}_2) &= \|\boldsymbol{S}(\boldsymbol{P}_1 - \boldsymbol{P}_2)\|_2 = \|(\boldsymbol{S}\boldsymbol{P}_1 - \boldsymbol{S}\boldsymbol{P}_3) + (\boldsymbol{S}\boldsymbol{P}_3 - \boldsymbol{S}\boldsymbol{P}_2)\|_2 \\ &\quad \text{(via the \textit{triangular inequality} concerning the \textit{Euclidean distance})} \\ &\leqslant \|\boldsymbol{S}\boldsymbol{P}_1 - \boldsymbol{S}\boldsymbol{P}_3\|_2 + \|\boldsymbol{S}\boldsymbol{P}_3 - \boldsymbol{S}\boldsymbol{P}_2\|_2 \\ &= d_{\boldsymbol{M}}(\boldsymbol{P}_1, \boldsymbol{P}_3) + d_{\boldsymbol{M}}(\boldsymbol{P}_3, \boldsymbol{P}_2). \end{aligned}$$

The triangular inequality holds and the proof is done.

3.2.4 Distance, Area, Volume, and Fractal Measures

The distance can be conceptually regarded as the most fundamental one among common measures. One may somehow axiomatically treat the measure of *area* as product of two (orthogonal) distances. Besides being computationally pragmatic for human activities, such axiomatic treatment of area is also theoretically useful for theorem proving. For example, to prove the *Gou Gu* theorem or *Pythagorean* theorem, one may construct the figure illustrated in Fig. 3.8 and take advantage of area.

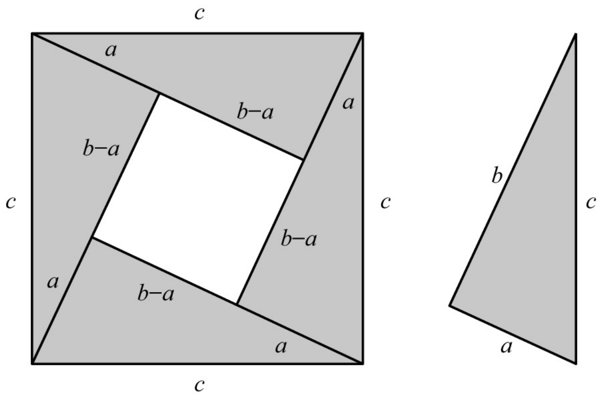


Fig. 3.8 Gou Gu theorem proved via area

The large square consists of the internal small square and four identical right-angle triangles, so

$$\begin{aligned} \text{area of large square} &= \text{area of small square} + 4 \times \text{area of right} - \text{angle triangle} \\ \Longleftrightarrow c^2 &= (b-a)^2 + 4 \times (ab/2) = a^2 - 2ab + b^2 + 2ab = a^2 + b^2 \end{aligned}$$

which conveniently proves the Gou Gu theorem.

For another example, to prove the *Menelaus theorem* and the *Ceva* theorem applied to triangles illustrated in Fig. 3.9, one can also take advantage of area.

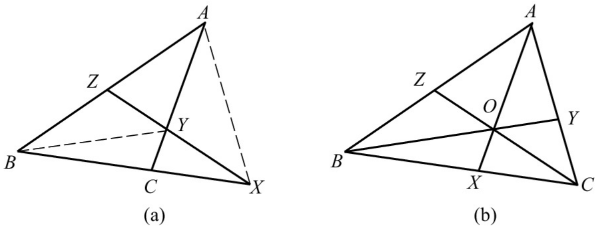


Fig. 3.9 Menelaus theorem (**a**) and Ceva theorem (**b**) proved via area

Consider the triangle *A*-*B*-*C* intercepted by the line *X*-*Y*-*Z*, as illustrated in Fig. 3.9a. For the triangle *A*-*B*-*C*, note that

$$\begin{aligned} \frac{|AZ|}{|ZB|} &= \frac{\text{area of } \Delta AYX}{\text{area of } \Delta BYX}, \\ \frac{|BX|}{|XC|} &= \frac{\text{area of } \Delta BYX}{\text{area of } \Delta CYX}, \\ \frac{|CY|}{|YA|} &= \frac{\text{area of } \Delta CYX}{\text{area of } \Delta AYX}. \end{aligned}$$

Multiply both sides of above equalities and obtain

$$\frac{|AZ|}{|ZB|} \cdot \frac{|BX|}{|XC|} \cdot \frac{|CY|}{|YA|} = 1$$

which conveniently proves the Menelaus theorem. Then consider the triangle *A*-*B*-*C* with lines *A*-*X*, *B*-*Y*, and *C*-*Z* coinciding at *O*, as illustrated in Fig. 3.9b. For the triangle *A*-*B*-*C*, note that

$$\begin{aligned} \frac{|AZ|}{|ZB|} &= \frac{\text{area of } \Delta AOC}{\text{area of } \Delta BOC}, \\ \frac{|BX|}{|XC|} &= \frac{\text{area of } \Delta AOB}{\text{area of } \Delta AOC}, \\ \frac{|CY|}{|YA|} &= \frac{\text{area of } \Delta BOC}{\text{area of } \Delta AOB}. \end{aligned}$$

Multiply both sides of above equalities and obtain

$$\frac{|AZ|}{|ZB|} \cdot \frac{|BX|}{|XC|} \cdot \frac{|CY|}{|YA|} = 1$$

which conveniently proves the Ceva theorem.

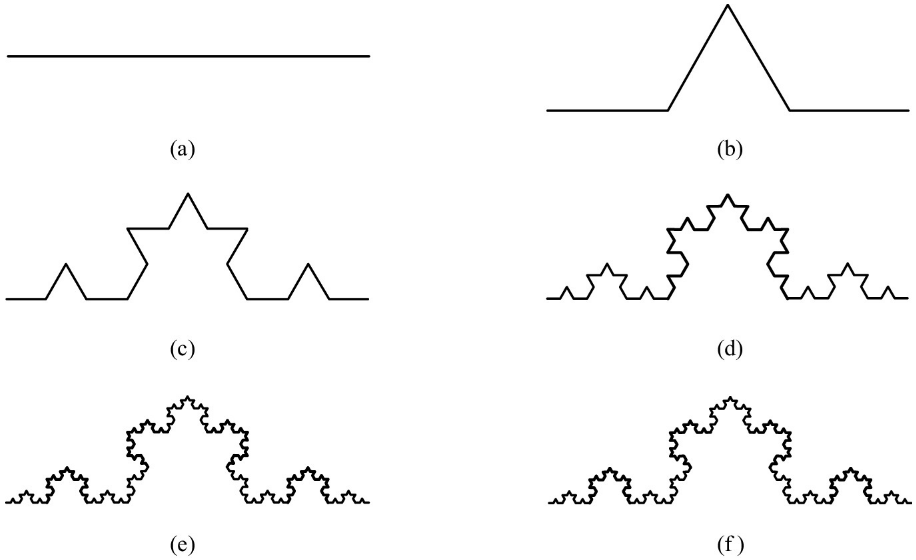
Similar to axiomatic treatment of area, one may somehow axiomatically treat the measure of *volume* as product of three distances. More generally, one may imagine some abstract high-dimensional measures as products of four distances, five distances, and so on, as formalized in (3.33):

$$m_{\boldsymbol{E}} \propto d_{\boldsymbol{E}}^{n} \quad (n \in \{0, 1, 2, 3, \cdots\}), \tag{3.33}$$

where $\boldsymbol{E}$ denotes a geometric entity, $m_{\boldsymbol{E}}$ denotes certain measure of $\boldsymbol{E}$, $d_{\boldsymbol{E}}$ denotes certain distance property such as the diameter of $\boldsymbol{E}$, and n denotes the dimension of $\boldsymbol{E}$. Further clarification of (3.33) will be given latter. **Fractals**

Besides geometric entities that have measures equivalent to products of integer number of distances, geometric entities that have measures equivalent to products of non-integer number of distances also exist *objectively* in the world. For example, the snow contour illustrated in Fig. 3.10 belongs to such kind of objective existence.

Necessary explanations hover over the snow contour illustrated in Fig. 3.10. In fact, due to inherent limitations of humans' cognition ability, people are unable to directly obtain the exact snow contour, but can indirectly understand (imagine) it as the limit of a sequence of quasi-snow contours obtained iteratively via self-similar transform .[7]



(a) (b) (c) (d) (e) (f)

Fig. 3.10 Snow fractal: refinement via iterative fractal transform

More specifically, start from a line segment as illustrated in Fig. 3.10a, transform it to a ___Λ___ shape as illustrated in Fig. 3.10b, namely divide the line segment equally into three small line segments, keep the two small line segments on both sides unchanged, and replace the small line segment in the middle by two identical new ones that still concatenate the two small line segments on both sides. The effect of transforming the line segment to the corresponding ___Λ___ shape is illustrated by the change from the sub-figure(a) to the sub-figure(b) in Fig. 3.10.

After the transform, the newly-obtained ___Λ___ shape consists of four small line segments, with one more than the original line segment that consists of three small line segments. Continue to perform such self-similar transform on the newly-obtained ___Λ___ shape. In other words, for each small line segment of the ___Λ___ shape, transform it to a corresponding ___Λ___ shape that consists of four small-small line segments. So for the ___Λ___ shape, after such self-similar transform of all its four small line segments, it becomes a ___Λ___ shape of four small ___Λ___ shapes. The effect of such self-similar transform is illustrated by the change from the sub-figure(b) to the sub-figure(c) in Fig. 3.10.

Repeat the self-similar transform iteratively to further obtain a ___Λ___ shape of four small ___Λ___ shapes of small-small ___Λ___ shapes as illustrated in Fig. 3.10(d), a ___Λ___ shape of four small ___Λ___ shapes of small-small ___Λ___ shapes of small-small-small ___Λ___ shapes as illustrated in Fig. 3.10(e), and so on infinitely.

After each round of the self-similar transform, a more refined quasi-snow contour will be generated. Starting from the original line segment, the intermediate quasi-snow contours obtained after one, two, three, four, and five rounds of the self-similar transform are illustrated respectively in Fig. 3.10b–f. Denote the quasi-snow contour obtained after k rounds of the self-similar transform as $\boldsymbol{S}_k$. By above repetitive process, one can obtain a sequence of quasi-snow contours

$$\boldsymbol{S}_0, \quad \boldsymbol{S}_1, \quad \boldsymbol{S}_2, \quad \boldsymbol{S}_3, \quad \boldsymbol{S}_4, \quad \boldsymbol{S}_5, \quad \cdots$$

which become more and more refined and will converge to a limit snow contour [8]

$$\boldsymbol{S}_\infty \equiv \lim_{k\to\infty} \boldsymbol{S}_k.$$

Then the limit snow contour $\boldsymbol{S}_\infty$, which exists objectively or is logically realizable,[9] is the snow contour under discussion. It is actually a ___Λ___ shape of four small copies of itself.

The snow contour $\boldsymbol{S}_\infty$ is also known as the *Koch curve* (originally *la courbe de von Koch* in French), named after von Koch who made such curve debut [13]. It is a representative of what people know as **fractals** since 1970s [14, 15]. Fractals have a number of fascinating properties: First, examined at no matter what an arbitrarily infinitesimal scale, they always have delicate details of their inherent characteristics. Second, conventional concepts and tools in traditional geometry, especially traditional measures such as the *Jordan measure* and the *Lebesgue measure* [16], are not applicable to them. Third, they have their own regularities of generating themselves, via self-similar transforms, stochastic self-similar transforms, self-affine transforms, stochastic self-affine transforms, and other transforms (for simplicity, one may refer to the various transforms together as **fractal transforms**). Fourth, they usually have no place differentiable, be them continuous or not—Among the four properties described here, this last property is most related to the motivation of creating the new term "fractal" from its Latin origin "fractus" which means broken [14].

Since the snow contour $\boldsymbol{S}_\infty$ consists of parts that are similar to itself and differ from itself only by scaling factors, it is also a representative of *scaling fractals*, in contrast with *non-scaling fractals*. Matlab code for generating the snow fractal illustrated in Fig. 3.10 is given as follows.

GenerateSnow.m

```
%% Generate the snow fractal
% fN: Number of fractal transform
% s: Current snow fractal [[x1;y1], [x2;y2], ..., [xN;yN]]
% sr: Refined snow fractal via iterative fractal transform
function sr = GenerateSnow(fN, s)
    if (fN==0) sr = s; return; end
    n = size(s,2)-1; sr = zeros(2,4*n+1);
    for k=1:n
        sr(:,4*k-3:4*k+1) = SnowFractalTransform(s(:,[k,k+1]));
    end
    sr = GenerateSnow(fN-1, sr);

    function segnew = SnowFractalTransform(seg)
        A = seg(:,1); B = seg(:,2); C = (2*A+B)/3; D = (A+2*B)
    /3;
        E = C+[cos(pi/3), -sin(pi/3); sin(pi/3), cos(pi/3)]*(D-C
    );
        segnew = [A,C,E,D,B];
    end
end
```

Non-integer (fractal) Dimensions

The snow fractal seems to be one-dimensional line, so try to compute its one-dimensional measure m_L namely length. Suppose the length of the original line segment $\boldsymbol{S}_0$ is $m_{L,\boldsymbol{S}_0} = L$. Consider $\boldsymbol{S}_k$, which consists of 4^k small line segments of length $L/3^k$, so

$$m_{L,\boldsymbol{S}_k} = 4^k \cdot \frac{L}{3^k} = \left(\frac{4}{3}\right)^k L$$

and

$$m_{L,\boldsymbol{S}_\infty} = \lim_{k\to\infty} m_{L,\boldsymbol{S}_k} = \lim_{k\to\infty} \left(\frac{4}{3}\right)^k L = \infty.$$

The infinite length of the snow fractal implies that it should be higher than one-dimensional. Then try to compute its two-dimensional measure m_A namely area. Still consider $\boldsymbol{S}_k$ and cover each of its 4^k small line segments by a small square of side length $L/3^k$, so

$$m_{A,\boldsymbol{S}_k} \leqslant 4^k \cdot \left(\tfrac{L}{3^k}\right)^2 = \left(\tfrac{4}{9}\right)^k L^2$$

and

$$\begin{aligned} m_{A,\boldsymbol{S}_\infty} = \lim_{k\to\infty} m_{A,\boldsymbol{S}_k} \leqslant \lim_{k\to\infty} \left(\tfrac{4}{9}\right)^k L^2 &= 0 \\ \Rightarrow m_{A,\boldsymbol{S}_\infty} &= 0. \end{aligned}$$

The zero area of the snow fractal implies that it should be lower than two-dimensional.

Recall the author's previous statement that the snow fractal belongs to geometric entities that have measures equivalent to products of non-integer number of distances. The above analysis reflects right that there is no choice but a non-integer number between one and two for the dimension of the snow fractal. To see the point better and even reveal the actual non-integer dimension of the snow fractal, further clarification of (3.33) is needed.

The formalism of (3.33) can be interpreted in the following way: Given a geometric entity $\boldsymbol{E}$, if and only if it is really n-dimensional, then the ratio between $m_{\boldsymbol{E}}$ and $d_{\boldsymbol{E}}^n$ will always be the same no matter how one scales $\boldsymbol{E}$. For example, given a (two-dimensional) triangle $\boldsymbol{T}$ with diameter $d_{\boldsymbol{T}}$ and area measure $m_{\boldsymbol{T}}$. If the triangle is scaled by a factor λ, then the scaled triangle $\lambda\boldsymbol{T}$ will have a new diameter $d_{\lambda\boldsymbol{T}} = \lambda d_{\boldsymbol{T}}$ and a new area measure $m_{\lambda\boldsymbol{T}} = \lambda^2 m_{\boldsymbol{T}}$, but the ratio between $m_{\lambda\boldsymbol{T}}$ and $d_{\lambda\boldsymbol{T}}^2$ (where $n = 2$ for this triangle example) is

$$\tfrac{m_{\lambda\boldsymbol{T}}}{d_{\lambda\boldsymbol{T}}^2} = \tfrac{\lambda^2 m_{\boldsymbol{T}}}{(\lambda d_{\boldsymbol{T}})^2} = \tfrac{m_{\boldsymbol{T}}}{d_{\boldsymbol{T}}^2},$$

the same to the original ratio between $m_{\boldsymbol{T}}$ and $d_{\boldsymbol{T}}^2$. So the triangle is right $n = 2$ dimensional.

If one takes a dimension $n > 2$ such as $n = 3$, then the new ratio associated with the scaled triangle $\lambda\boldsymbol{T}$ becomes

$$\tfrac{m_{\lambda\boldsymbol{T}}}{d_{\lambda\boldsymbol{T}}^3} = \tfrac{\lambda^2 m_{\boldsymbol{T}}}{(\lambda d_{\boldsymbol{T}})^3} = \tfrac{1}{\lambda}\tfrac{m_{\boldsymbol{T}}}{d_{\boldsymbol{T}}^3},$$

which decreases as λ increases. In contrast, if one takes a dimension $n < 2$ such as $n = 1$, then the new ratio associated with the scaled triangle $\lambda\boldsymbol{T}$ becomes

$$\tfrac{m_{\lambda\boldsymbol{T}}}{d_{\lambda\boldsymbol{T}}} = \tfrac{\lambda^2 m_{\boldsymbol{T}}}{\lambda d_{\boldsymbol{T}}} = \lambda\tfrac{m_{\boldsymbol{T}}}{d_{\boldsymbol{T}}},$$

which increases as λ increases. In both cases, the new ratio associated with the scaled triangle $\lambda\boldsymbol{T}$ has changed.

Based on above interpretation, one can analyse the dimension s of the snow fractal. On one hand, apply (3.33) to the entire snow fractal $\boldsymbol{S}_\infty$ of diameter L and obtain

$$m_{\boldsymbol{S}_\infty} = c\, d_{\boldsymbol{S}_\infty}^s = c\, L^s, \tag{3.34}$$

where c denotes certain constant associated with the snow fractal. On the other hand, since the snow fractal $\boldsymbol{S}_\infty$ is actually a __Λ__ shape of four small copies of itself, apply (3.33) to each small copy of diameter $L/3$ and obtain

$$m_{\boldsymbol{S}_\infty} = 4 \cdot c\left(\frac{L}{3}\right)^s = \frac{4}{3^s} c\ L^s. \tag{3.35}$$

Compare (3.34) and (3.35) to establish the equality

$$c\ L^s = \frac{4}{3^s} c\ L^s \Longleftrightarrow 1 = \frac{4}{3^s},$$

which implies

$$s = \frac{\log 4}{\log 3} \approx 1.262. \tag{3.36}$$

In a similar way based on (3.33), one can also analyse the dimension of the Cantor fractal, i.e. the *Cantor set* [17] illustrated in Fig. 3.11. Start from a line segment and repeat the demonstrated self-similar transform iteratively to obtain the Cantor fractal. The self-similar transform consists in dividing each composing line segment equally into three smaller line segments and discarding the middle one.[10]

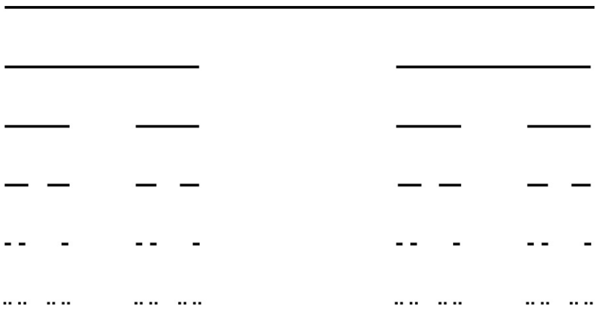


Fig. 3.11 Cantor set (fractal): refinement via iterative fractal transform

Suppose the length of the original line segment is L, namely the diameter of the entire Cantor fractal is L. On one hand, apply (3.33) to the entire Cantor fractal $\boldsymbol{C}_\infty$ of diameter L and obtain

$$m_{\boldsymbol{C}_\infty} = c\ d^s_{\boldsymbol{C}_\infty} = c\ L^s, \tag{3.37}$$

$$m_{\boldsymbol{C}_\infty} = 2 \cdot c\left(\frac{L}{3}\right)^s = \frac{2}{3^s} c\ L^s. \tag{3.38}$$

Compare (3.37) and (3.38) to establish the equality

$$c\ L^s = \frac{2}{3^s} c\ L^s \Longleftrightarrow 1 = \frac{2}{3^s},$$

which implies

$$s = \frac{\log 2}{\log 3} \approx 0.631. \tag{3.39}$$

Such non-integer dimensions of fractals, for example, the snow fractal's non-integer dimension derived in (3.36) and the Cantor fractal's non-integer dimension derived in

(3.39), will be called **fractal dimensions**.

Hausdorff Measure and Dimension

The convenient way of deriving fractal dimensions via (3.33), as demonstrated by derivation of (3.36) and (3.39), is not generally applicable. In fact, it is indeed applicable to strictly self-similar fractals, but will become invalid if one simply adds a bit of stochastic “ingredient” in fractals, not to say for an ocean of cases where strict self-similarity of fractals is violated.

For example, for generation of the snow fractal, one may change just one point of its fractal transform: Instead of fixing the wedge direction, i.e. direction of “Λ” in the ___Λ___ shape, set the wedge direction randomly between the up direction and the down direction to form the ___Λ___ shape or the inverted ___Λ___ shape, as illustrated in Fig. 3.12a, b respectively. The process of generating the stochastic snow fractal (or the stochastic Koch curve) in such stochastic way is demonstrated sequentially in Fig. 3.12b–f. Matlab code for generating the stochastic snow fractal illustrated in Fig. 3.12 is given as follows.

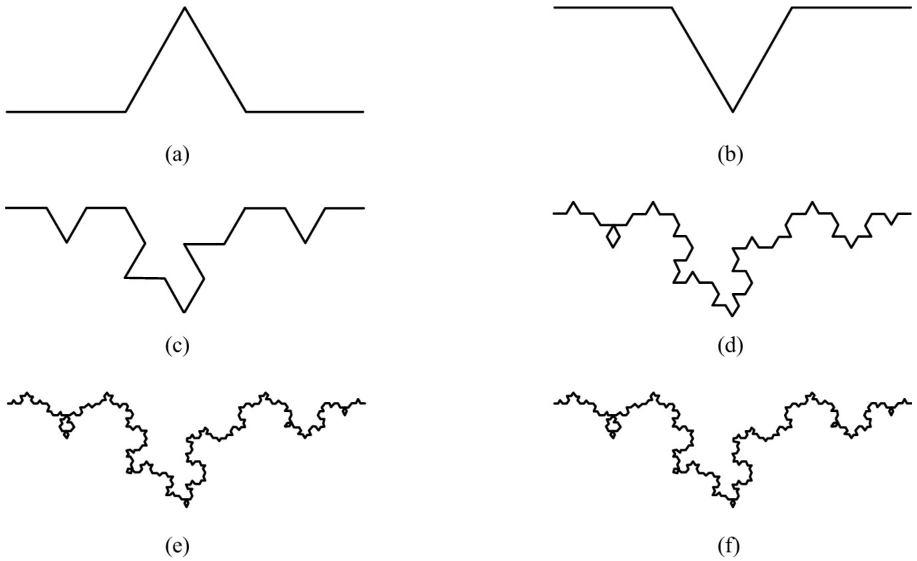

(a) (b) (c) (d) (e) (f)

Fig. 3.12 Stochastic snow fractal: refinement via iterative stochastic fractal transform

StochasticGenerateSnow.m

```
%% Generate the stochastic snow fractal
% d: 1 for up wedge direction; 0 for down wedge direction
function segnew = StochasticSnowFractalTransform(seg, d)
    A = seg(:,1); B = seg(:,2); C = (2*A+B)/3; D = (A+2*B)/3;
    if(d) a = pi/3; else a = -pi/3; end
    E = C+[cos(a), -sin(a); sin(a), cos(a)]*(D-C);
    segnew = [A,C,E,D,B];
end
% fN: Number of fractal transform
% s: Current snow fractal [[x1;y1], [x2;y2], ..., [xN;yN]]
% sr: Refined snow fractal via iterative fractal transform
function sr = StochasticGenerateSnow(fN, s)
    if (fN==0) sr = s; return; end
    n = size(s,2)-1; sr = zeros(2,4*n+1);
    for k=1:n
        d = round(rand()); % Set the wedge direction randomly
        sr(:,4*k-3:4*k+1) = StochasticSnowFractalTransform(s(:,[
    k,k+1]),d);
    end
    sr = StochasticGenerateSnow(fN-1, sr);
end
```

Then how to analyse the dimension of the stochastic snow fractal and even measure it at its corresponding dimension? The answer to the question involves the **Hausdorff measure** [18] defined as [11]

$$m_{\boldsymbol{E}}^{H,s} \equiv \mathop{\mathrm{liminf}}_{\delta\to 0} \{c_s \sum_{i=1}^{\infty} d_{\boldsymbol{K}_i}^{s} \mid \boldsymbol{E} \subseteq \cup_{i=1}^{\infty} \boldsymbol{K}_i, d_{\boldsymbol{K}_i} < \delta\}, \tag{3.40}$$

where $0 \leqslant s < \infty$ denotes the dimension under consideration, the union of $\{\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3, \cdots\}$ denotes a *countable* open-set cover of $\boldsymbol{E}$, and c_s denotes the “volume” of the s-dimensional unit-diameter sphere

$$c_s = \frac{\pi^{\frac{s}{2}}}{\Gamma\left(\frac{s}{2}+1\right)} \left(\frac{1}{2}\right)^{s}.$$

Since c_s is constant for all s-dimensional geometric entities and it is normally meaningless to compare measures of different dimensions, one can fairly omit c_s in definition of (3.40) as

$$m_{\boldsymbol{E}}^{H,s} \equiv \lim_{\delta\to 0}\inf \left\{ \sum_{i=1}^{\infty} d_{\boldsymbol{K}_i}^{s} \mid \boldsymbol{E} \subseteq \cup_{i=1}^{\infty} \boldsymbol{K}_i, d_{\boldsymbol{K}_i} < \delta \right\}. \tag{3.41}$$

Given a geometric entity $\boldsymbol{E}$, as the dimension s increases, its Hausdorff measure $m_{\boldsymbol{E}}^{H,s}$ will drop to 0 when s passes beyond certain dimension, namely the infimum of all s whose associated Hausdorff measures $m_{\boldsymbol{E}}^{H,s}$ are zero.[12] The **Hausdorff dimension** [18] is defined right as such infimum

$$s_{\boldsymbol{E}}^{H} \equiv \inf \{s \mid m_{\boldsymbol{E}}^{H,s} = 0\}. \tag{3.42}$$

Return to analysis of the dimension of the stochastic snow fractal and still abuse $\boldsymbol{S}_k$ to denote the stochastic quasi-snow contour obtained after k rounds of the stochastic fractal transform. Suppose the length of the original line segment $\boldsymbol{S}_0$ is $m_{L,\boldsymbol{S}_0} = L$. For $\boldsymbol{S}_k$, despite stochastic irregularity of its wedge directions, it definitely consists of 4^k small line segments of length $L/3^k$. Consider a special open-set cover of $\boldsymbol{S}_k$, namely a union of 4^k disks of diameter $L/3^k$, each of which covers exactly a small line segment in $\boldsymbol{S}_k$. The special open-set cover of $\boldsymbol{S}_k$ is a countable open-set cover of the stochastic snow fractal $\boldsymbol{S}_\infty$ as well.[13] For the special open-set cover of $\boldsymbol{S}_k$, denote the 4^k disks of diameter $L/3^k$ as

$$\boldsymbol{D}_{k,1}, \quad \boldsymbol{D}_{k,2}, \quad \cdots \quad , \quad \boldsymbol{D}_{k,4^k-1}, \quad \boldsymbol{D}_{k,4^k}.$$

So apply (3.41) and obtain

$$\begin{aligned} m^{H,s}_{\boldsymbol{S}_\infty} &\equiv \lim\nolimits_{\delta\to 0}\inf\{\text{measures of all open} - \text{set covers with open} - \text{set diameter} < \delta\} \\ &\leqslant \lim\nolimits_{\delta\to 0}\{\text{measures of special open} - \text{set covers with open} - \text{set diameter} < \delta\} \\ &= \lim_{k\to\infty}\sum_{i=1}^{4^k} d^s_{\boldsymbol{D}_{k,i}} = \lim_{k\to\infty} 4^k \cdot \left(\frac{L}{3^k}\right)^s = L^s \lim_{k\to\infty}\left(\frac{4}{3^s}\right)^k. \end{aligned}$$

(3.

If the dimension $s >\log 4/\log 3 \approx 1.262$, then $4/3^s < 1$ and (3.43) implies

$$m^{H,s}_{\boldsymbol{S}_\infty} \leqslant L^s \lim\nolimits_{k\to\infty}\left(\tfrac{4}{3^s}\right)^k = 0. \tag{3.44}$$

If the dimension $s <\log 4/\log 3 \approx 1.262$, then $4/3^s > 1$ and the author will show

$$m^{H,s}_{\boldsymbol{S}_\infty} = \infty \Longleftrightarrow \lim_{\delta\to 0}\sum_{i=1}^{\infty} d^s_{\boldsymbol{K}_i}\big|_{\boldsymbol{S}_\infty \subseteq \cup_{i=1}^{\infty}\boldsymbol{K}_i, d_{\boldsymbol{K}_i}<\delta} = \infty. \tag{3.45}$$

In other words, the author will show measures of all open-set covers with open-set diameter arbitrarily infinitesimal are inevitably infinite.

Before proving (3.45), consider the special open-set cover of $\boldsymbol{S}_k$ namely

$$\boldsymbol{D}_{k,1}, \quad \boldsymbol{D}_{k,2}, \quad \cdots \quad , \quad \boldsymbol{D}_{k,4^k-1}, \quad \boldsymbol{D}_{k,4^k}$$

and compute their associated measure with $s <\log 4/\log 3$

$$\lim_{k\to\infty}\sum_{i=1}^{4^k} d^s_{\boldsymbol{D}_{k,i}} = \lim_{k\to\infty} 4^k \cdot \left(\frac{L}{3^k}\right)^s = L^s \lim_{k\to\infty}\left(\frac{4}{3^s}\right)^k = \infty. \tag{3.46}$$

The analysis given in (3.46) tells that if an open-set cover consists of open sets of the same diameter, then their associated measure will be infinite as the open-set diameter approaches zero. Motivated by (3.46), for an arbitrary open-set cover, the open sets are assigned to different groups, each of which consists of open sets of roughly the same diameter.

Proof Bear $s <\log 4/\log 3$ and hence $4/3^s > 1$ in mind by default. Given an arbitrary open-set cover $\{\boldsymbol{K}_i\}$ of $\boldsymbol{S}_\infty$, let A_k denote the set of open sets of diameters within the interval $I_k \equiv (L/3^{k+1}, L/3^k]$, let $|A_k|$ denote its cardinality, and let r_k denote the ratio of the $\boldsymbol{S}_\infty$ part covered by A_k with respect to the whole $\boldsymbol{S}_\infty$. It is not difficult to verify

$$r_k < \tfrac{2|A_k|}{4^k}$$

and hence

$$|A_k| > \tfrac{4^k}{2} r_k.$$

Since $\{\boldsymbol{K}_i\}$ is a cover of $\boldsymbol{S}_\infty$, one has

$$\sum_{k=k_{\min}}^{\infty} r_k \geqslant 1$$

and

$$\begin{aligned}\lim_{\delta \to 0} \sum_{i=1}^{\infty} d^s_{\boldsymbol{K}_i} &> \lim_{k_{\min} \to \infty} \sum_{k=k_{\min}}^{\infty} |A_k| \left(\frac{L}{3^{k+1}}\right)^s > \frac{1}{2}\left(\frac{L}{3}\right)^s \lim_{k_{\min} \to \infty} \sum_{k=k_{\min}}^{\infty} r_k \left(\frac{4}{3^s}\right)^k \\ &> \frac{1}{2}\left(\frac{L}{3}\right)^s \lim_{k_{\min} \to \infty} \sum_{k=k_{\min}}^{\infty} r_k \left(\frac{4}{3^s}\right)^{k_{\min}} \geqslant \frac{1}{2}\left(\frac{L}{3}\right)^s \lim_{k_{\min} \to \infty} \left(\frac{4}{3^s}\right)^{k_{\min}} \\ &= \infty.\end{aligned}$$

The proof is done. □

Associate (3.44) and (3.45) to derive the Hausdorff dimension of the stochastic snow fractal via (3.42) as

$$s^H_{\boldsymbol{S}_\infty} \equiv \inf \{s \mid m^{H,s}_{\boldsymbol{S}_\infty} = 0\} = \tfrac{\log 4}{\log 3} \approx 1.262, \tag{3.47}$$

the same to the dimension of the snow fractal given in (3.36). In fact, what (3.36) describes is right the Hausdorff dimension of the snow fractal. Similarly, what (3.39) describes is right the Hausdorff dimension of the Cantor fractal.

From Theoretical Abstraction to Practical Tractability

Fractal dimensions and measures are indeed beyond humans' intuitive understanding, yet their *objective existence* have been reflected and analysed indirectly. Such objective existence is like that of wind (air movement), which one can never see directly but can know indirectly according to a number of natural phenomena such as movement of tree leaves. Their objective existence is even more like objective existence of curved space-time which is also beyond intuitive understanding.

Theoretical abstraction is inevitably needed to describe and analyse fractal dimensions and measures . Readers might think the Hausdorff measure is already very abstract and general. In fact, it is just a special instantiation of the more abstract and general spirit of **Carathéodory construction** [19], yet the author refrains from advancing towards deeper abstraction and generalization, which is after all not so necessary for practical applications in the context of computer vision.

Even with the Hausdorff measure , the author does not tend to follow their definitions strictly but instead follow their simplified variants for practical tractability or at least for application convenience. More specifically, cover a geometric entity with open sets (say "boxes") of the same metric when analysing its measure. For example, as presented previously, apply the special open-set cover to the stochastic snow fractal. The fractal dimension computed in such way is called the *Minkowski dimension* also known as the *box-packing dimension* [20] or *box-counting dimension* [21]. The author has already demonstrated via (3.44) and (3.46) how the box-packing dimension of the stochastic snow fractal can be conveniently estimated.

Given a geometric entity $\boldsymbol{E}$, the box-packing dimension can be even more conveniently estimated via

$$s_{\boldsymbol{E}}^{B} = \lim_{\delta \to 0} \frac{\log n(\boldsymbol{E}, \delta)}{\log(1/\delta)}, \tag{3.48}$$

where δ denotes the diameter of the boxes and $n(\boldsymbol{E}, \delta)$ denotes the number of boxes that cover the geometric entity $\boldsymbol{E}$. For example, apply (3.48) to the snow fractal or the stochastic snow fractal and obtain

$$s_{\boldsymbol{S}_\infty}^{B} = \lim_{k \to \infty} \frac{\log 4^k}{\log(3^k/L)} = \lim_{k \to \infty} \frac{k \log 4}{k \log 3 - \log L} = \frac{\log 4}{\log 3} \approx 1.262.$$

Apply (3.48) to the Cantor fractal and obtain

$$s_{\boldsymbol{C}_\infty}^{B} = \lim_{k \to \infty} \frac{\log 2^k}{\log(3^k/L)} = \lim_{k \to \infty} \frac{k \log 2}{k \log 3 - \log L} = \frac{\log 2}{\log 3} \approx 0.631.$$

Besides, apply (3.48) to the Γ shape fractal [14] illustrated in Fig. 3.13 and obtain

$$s_{\boldsymbol{\Gamma}_\infty}^{B} = \lim_{k \to \infty} \frac{\log 3^k}{\log(2^k/L)} = \lim_{k \to \infty} \frac{k \log 3}{k \log 2 - \log L} = \frac{\log 3}{\log 2} \approx 1.585.$$

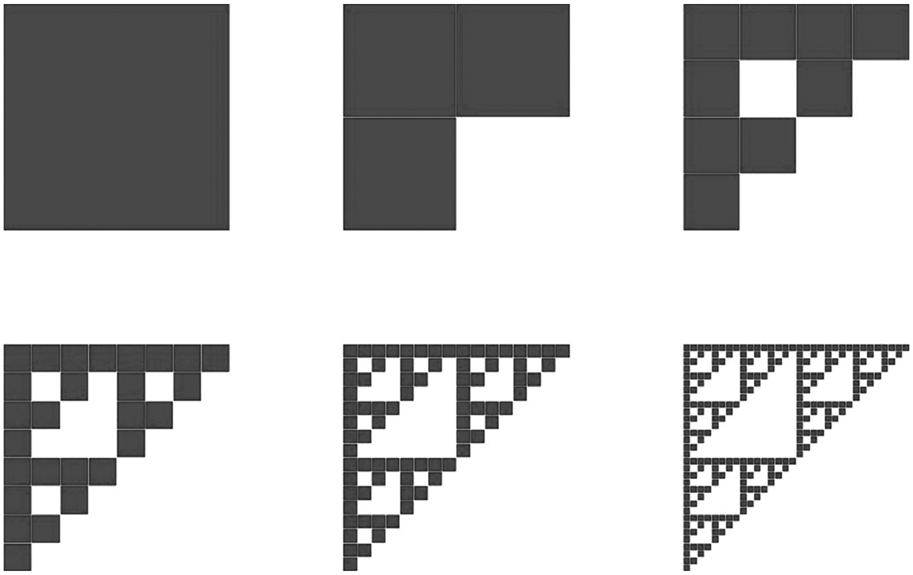


Fig. 3.13 Γ shape fractal: refinement via iterative fractal transform

Awareness of objective existence of fractal measures and dimensions is important for some applications in the context of computer vision. Tractable techniques for computing fractal measures and dimensions can be applied to modelling and analysis of biological vision systems or biologically-inspired computer vision systems, of image compression quality, and of spatial-temporal varying illumination [23].

3.3 Graph

3.3.1 Topological Abstraction

In practical applications of computer vision, one may need to deal with *distances* and their derived measures which characterize *quantitative* properties of geometric entities. One may also need to deal with *topology* which characterizes *qualitative* properties of geometric entities. A representative method of topological abstraction for geometric entities is the graph method.

The debut of the concept **graph** is attributed to *Euler* [24], for solving the famous *seven bridges of Konigsberg* problem illustrated in Fig. 3.14. A **graph** is an abstract structure that consists of **vertices** denoted as

$$\boldsymbol{V} = \{V_1, V_2, \cdots, V_{|\boldsymbol{V}|}\}$$

and **edges** denoted as

$$\boldsymbol{E} = \{E_1, E_2, \cdots, E_{|\boldsymbol{E}|}\},$$

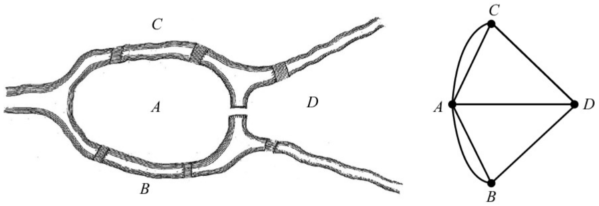


Fig. 3.14 Seven bridges of Konigsberg

where $|\boldsymbol{V}|$ denotes the number of vertices and $|\boldsymbol{E}|$ denotes the number of edges. The vertices represent key composing elements of a geometric entity under consideration in practical applications, whereas the edges represent relationships among the composing elements within the geometric entity.[15]

$$\left[\begin{array}{ccccc} & A & B & C & D \\ A & 0 & 2 & 2 & 1 \\ B & 2 & 0 & 0 & 1 \\ C & 2 & 0 & 0 & 1 \\ D & 1 & 1 & 1 & 0 \end{array}\right]$$

or simply as[16]

$$\begin{bmatrix} 0 & 2 & 2 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

if one gives vertices $\{A, B, C, D\}$ a default order or indexing as $\{1, 2, 3, 4\}$. One may also adopt an adjacency list representation for the seven-bridge graph as

$$\begin{aligned}
&A \to B \to B \to C \to C \to D \\
&B \to A \to A \to D \\
&C \to A \to A \to D \\
&D \to A \to B \to C
\end{aligned}$$

where each vertex links to its neighbour vertices consecutively.

A graph is called **dense** if $|\boldsymbol{E}| = O(|\boldsymbol{V}|^2)$. Intuitively speaking, in a graph, if on average each vertex connects to $O(|\boldsymbol{V}|)$ other vertices especially to almost all other vertices, then the graph is dense. A dense graph example is a generic group of points considered in the context of closest point problems, where each pair of points has a mutual distance and hence an implicit edge between them.

A graph is called **sparse** if $|\boldsymbol{E}| = O(|\boldsymbol{V}|)$. Intuitively speaking, in a graph, if on average each vertex connects to $O(1)$ other vertices especially to few number of other vertices, then the graph is sparse. A sparse graph example is a road map. Another sparse graph example is a planar straight-line graph [25].

The adjacency matrix representation is usually more suitable to a dense graph, whereas the adjacency list representation is usually more suitable to a sparse graph.

Undirected and Directed Graphs

A **directed graph** or **digraph** is such a graph that its edges have directions. More specifically, each edge has a direction indicating from which vertex and to which vertex the edge points or connects. A vertex may be connected to another vertex by a directed edge, whereas this other vertex is not connected to the vertex by any directed edge. In a directed graph, a pair of mutually connected vertices has two edges connecting them: one connecting from the first vertex to the second vertex, and the other connecting from the second vertex to the first vertex.

An **undirected graph** is such a graph that its edges are always bidirectional. In other words, if a vertex is connected to another vertex by an edge, then this other vertex must also be connected back to the vertex by the same edge. An undirected graph can be treated as a directed graph, if each edge is treated as two directed edges that associate the same pair of vertices but have opposite directions. In this sense, directed graphs are more general conceptually and it seems that one had better treat graphs as directed graphs by default.

However, undirected graphs tend to be more intuitive and natural to humans' daily-life understanding. Take urban traffic networks as example, it is true that people do occasionally encounter one-way (unidirectional) roads which can be abstractly modelled as directed edges, yet most roads that people encounter in daily life are two-way (bidirectional) roads and hence it is natural for people not to distinguish between the opposite driving directions of the same road. For another example, when one can call a phone from another phone, it tends to be taken for granted that one can call this other phone back from the phone as well, though unidirectional calls do exist among special phones in reality. Therefore, without explicit mentioning of directed graphs, the author treats graphs as undirected graphs by default.

It is worth noting that the adjacency matrix of an undirected graph is a real-value symmetric matrix.

Laplacian Matrix

Given a graph with vertices

$$\boldsymbol{V} = \{V_1, V_2, \cdots, V_{|\boldsymbol{V}|}\}$$

and edges

$$\boldsymbol{E} = \{E_1, E_2, \cdots, E_{|\boldsymbol{E}|}\},$$

denote its $|\boldsymbol{V}|$-by-$|\boldsymbol{V}|$ adjacency matrix as

$$\boldsymbol{W} \equiv \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1|\boldsymbol{V}|} \\ {[3pt]} w_{12} & 0 & w_{23} & \cdots & w_{2|\boldsymbol{V}|} \\ {[3pt]} w_{13} & w_{23} & 0 & \cdots & w_{3|\boldsymbol{V}|} \\ {[3pt]} \vdots & \vdots & \vdots & \ddots & \vdots \\ {[3pt]} w_{1|\boldsymbol{V}|} & w_{2|\boldsymbol{V}|} & w_{3|\boldsymbol{V}|} & \cdots & 0 \end{bmatrix}$$

and its *diagonal matrix of weights* or *diagonal matrix of degrees* as

$$\begin{aligned} \boldsymbol{D} \quad &\equiv \begin{bmatrix} d_1 & & & & \\ {[3pt]} & d_2 & & & \\ {[3pt]} & & d_3 & & \\ {[2pt]} & & & \ddots & \\ {[2pt]} & & & & d_{|\boldsymbol{V}|} \end{bmatrix} \\ d_1 \quad &= w_{12} + w_{13} + \cdots + w_{1|\boldsymbol{V}|}, \\ d_2 \quad &= w_{12} + w_{23} + \cdots + w_{2|\boldsymbol{V}|}, \\ d_3 \quad &= w_{13} + w_{23} + \cdots + w_{3|\boldsymbol{V}|}, \\ \vdots \quad &= \quad \vdots \\ d_{|\boldsymbol{V}|} \quad &= w_{1|\boldsymbol{V}|} + w_{2|\boldsymbol{V}|} + w_{3|\boldsymbol{V}|} + \cdots + w_{|\boldsymbol{V}|-1,|\boldsymbol{V}|}. \end{aligned}$$

Each d_i in the diagonal matrix $\boldsymbol{D}$ equals the sum of all elements in the i-th row (or the i-th column) of the adjacency matrix $\boldsymbol{W}$.

The Laplacian matrix [26] of the graph is defined as

$$\boldsymbol{L} \equiv \boldsymbol{D} - \boldsymbol{W} = \begin{bmatrix} d_1 & -w_{12} & -w_{13} & \cdots & -w_{1|\boldsymbol{V}|} \\ {[3pt]} - w_{12} & d_2 & -w_{23} & \cdots & -w_{2|\boldsymbol{V}|} \\ {[3pt]} - w_{13} & -w_{23} & d_3 & \cdots & -w_{3|\boldsymbol{V}|} \\ {[3pt]} \vdots & \vdots & \vdots & \ddots & \vdots \\ {[3pt]} - w_{1|\boldsymbol{V}|} & -w_{2|\boldsymbol{V}|} & -w_{3|\boldsymbol{V}|} & \cdots & d_{|\boldsymbol{V}|} \end{bmatrix}. \tag{3.49}$$

The Laplacian matrix $\boldsymbol{L}$, according to its definition in (3.49), is a real-value symmetric matrix and hence its eigenvalues are all real. Besides, the diagonal elements of the Laplacian matrix are non-negative. Moreover, the Laplacian matrix is *diagonally*

dominant [17] and hence its eigenvalues are not only real but also non-negative, which can be easily seen according to the *Gersgorin theorem*.[18] So the Laplacian matrix $\boldsymbol{L}$ is positive semi-definite.

In fact, even without support from the Gersgorin theorem, the positive semi-definiteness of the Laplacian matrix can also be verified conveniently as follows

$$\begin{aligned} \boldsymbol{z}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{z} &= \sum_{i=1}^{|\boldsymbol{V}|}\sum_{j=1,j\neq i}^{|\boldsymbol{V}|} w_{ij}z_i^2 - 2\sum_{1\leqslant i<j\leqslant|\boldsymbol{V}|} w_{ij}z_iz_j \quad \left(\forall\ \boldsymbol{z}=\begin{bmatrix} z_1 & \cdots & z_{|\boldsymbol{V}|}\end{bmatrix}^{\mathrm{T}}\right) \\ &= \sum_{1\leqslant i<j\leqslant|\boldsymbol{V}|} w_{ij}(z_i-z_j)^2 \geqslant 0 \quad (\text{note each } w_{ij}=w_{ji}) \end{aligned} \tag{3.50}$$

where the equality holds if and only if

$$z_1 = z_2 = \cdots = z_{|\boldsymbol{V}|} \Longleftrightarrow \boldsymbol{z} \propto \mathbf{1} \equiv \begin{bmatrix} 1 & 1 & \cdots & 1\end{bmatrix}^{\mathrm{T}}. \tag{3.51}$$

A number of points about the Laplacian matrix $\boldsymbol{L}$ are conveyed by (3.50) and (3.51):

- The Laplacian matrix $\boldsymbol{L}$ is positive semi-definite.
- The Laplacian matrix $\boldsymbol{L}$ has one but only one eigenvector associated with the eigenvalue 0, namely the all-element-one vector, i.e. $\mathbf{1} \equiv \begin{bmatrix} 1 & 1 & \cdots & 1\end{bmatrix}^{\mathrm{T}}$.
- The Laplacian matrix $\boldsymbol{L}$ has a rank of $|\boldsymbol{V}| - 1$.

3.3.2 Planar Graphs and Euler Formula

A graph is called **planar** when its edges meet only at its vertices namely endpoints of its edges, or in other words when no edges intersect at places within any edge other than endpoints of the edge (including the requirement that a single edge does not intersect across itself at places other than its two endpoints)—Readers may realize that when talking about the planar graph, one assumes implicitly that the graph is embedded into the (two-dimensional) plane. The "planar" definition for graphs will be meaningless if they are not constrained to the plane, because any finite graph in three-dimensional space can easily satisfy the "planar" definition after at most an arbitrarily infinitesimal adjustment.[19]

A graph is called **simple** when it has no multiple edges joining the same pair of vertices. Apparently, the seven-bridge graph illustrated in Fig. 3.14 is not a simple graph. In following presentation, a planar graph is assumed simple by default, except when explicit mentioning of a non-simple planar graph is made.

A planar graph forms a subdivision of the plane. The parts of the plane segmented out by the edges of the graph are called **faces** or **regions** of the graph, denoted as

$$\boldsymbol{F} = \{F_1, F_2, \cdots, F_{|\boldsymbol{F}|}\},$$

where $|\boldsymbol{F}|$ denotes the number of faces. Two examples of planar graphs are illustrated in Fig. 3.15.

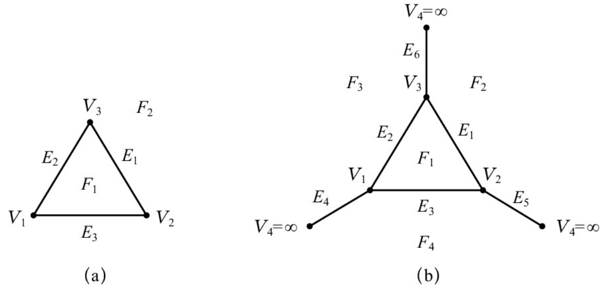


Fig. 3.15 Planar graphs **a** 3 vertices, 3 edges, and 2 faces; **b** 4 vertices, 6 edges, and 4 faces

The **Euler formula** states a well-known *invariance* of planar graphs [26]

$$|\boldsymbol{V}| - |\boldsymbol{E}| + |\boldsymbol{F}| = 2. \tag{3.52}$$

Consider the planar graph example illustrated in Fig. 3.15a, which has the vertex set, the edge set, and the face set

$$\begin{aligned} \boldsymbol{V} &= \{V_1, V_2, V_3\}, \\ \boldsymbol{E} &= \{E_1, E_2, E_3\}, \\ \boldsymbol{F} &= \{F_1, F_2\} \end{aligned}$$

and satisfies (3.52) as

$$|\boldsymbol{V}| - |\boldsymbol{E}| + |\boldsymbol{F}| = 3 - 3 + 2 = 2.$$

Consider the planar graph example illustrated in Fig. 3.15b, which has the vertex set,[20] the edge set, and the face set

$$\begin{aligned} \boldsymbol{V} &= \{V_1, V_2, V_3, V_4 = \infty\}, \\ \boldsymbol{E} &= \{E_1, E_2, E_3, E_4, E_5, E_6\}, \\ \boldsymbol{F} &= \{F_1, F_2, F_3, F_4\} \end{aligned}$$

and also satisfies (3.52) as

$$|\boldsymbol{V}| - |\boldsymbol{E}| + |\boldsymbol{F}| = 4 - 6 + 4 = 2.$$

The Euler formula can be proved via *mathematical induction*. The basic idea is that whenever an edge is removed from a graph, there will definitely be value reduction in $|\boldsymbol{V}| + |\boldsymbol{F}|$ the same to that in $|\boldsymbol{E}|$. If the sub-graph obtained after removal of the edge satisfies the invariance (3.52), then the original graph itself also satisfies (3.52).

Planar Graphs Are Sparse
Given a generic planar graph, if it has infinite-length edges, then imagine that all the infinite-length edges coincide at a virtual vertex at infinity. Suppose the faces F_1, F_2, ..., $F_{|\boldsymbol{F}|}$ have n_1 adjoining edges, n_2 adjoining edges, ..., $n_{|\boldsymbol{F}|}$ adjoining edges respectively. One has

$$2 \cdot |\boldsymbol{E}| = \sum_{i=1}^{|\boldsymbol{F}|} n_i.$$

Since each face has at least three adjoining edges, i.e. each $n_i \geqslant 3$, one has

$$2 \cdot |\boldsymbol{E}| \geqslant \sum_{i=1}^{|\boldsymbol{F}|} 3 = 3 \cdot |\boldsymbol{F}| \Longleftrightarrow |\boldsymbol{F}| \leqslant \frac{2}{3} |\boldsymbol{E}|.$$

Substitute above inequality into the Euler formula and obtain

$$\begin{aligned} 2 &= |\boldsymbol{V}| - |\boldsymbol{E}| + |\boldsymbol{F}| \leqslant |\boldsymbol{V}| - |\boldsymbol{E}| + \tfrac{2}{3} |\boldsymbol{E}| = |\boldsymbol{V}| - \tfrac{1}{3} |\boldsymbol{E}| \\ &\Rightarrow |\boldsymbol{E}| \leqslant 3(|\boldsymbol{V}| - 2) \Rightarrow |\boldsymbol{E}| = O(|\boldsymbol{V}|), \end{aligned}$$

namely the planar graph is sparse.

Besides, one further has

$$|\boldsymbol{F}| \leqslant \tfrac{2}{3}|\boldsymbol{E}| \leqslant \tfrac{2}{3} \cdot 3(|\boldsymbol{V}| - 2) = 2(|\boldsymbol{V}| - 2) \Rightarrow |\boldsymbol{F}| = O(|\boldsymbol{V}|).$$

So for a planar graph, both the edge number $|\boldsymbol{E}|$ and the face number $|\boldsymbol{F}|$ are of the same number order as the vertex number $|\boldsymbol{V}|$. This fact plays an important role in evaluating computational complexity of algorithms applied to planar graphs.

3.3.3 Planar Straight-Line Graphs

Planar straight-line graphs (PSLG) [25] are important geometric entities in computational geometry. A planar straight-line graph is a planar graph with all its edges being straight line segments (infinite-length edges within the planar straight-line graph are treated as straight line segments between normal vertices and the virtual vertex at infinity). Since planar graphs are sparse, planar straight-line graphs are also sparse.

It is proved that on the plane every planar graph (assumed simple by default) is homeomorphic to a planar straight-line graph [25]. So one can fairly use planar straight-line graphs to represent planar graphs while keeping topological essence of planar graphs. Application of planar straight-line graphs instead of general planar graphs facilitates topological analysis as well as quantitative geometric computations.

PSLG Representation

A generic graph $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}\}$ consists of the vertex set

$$\boldsymbol{V} = \{V_1, V_2, \cdots, V_{|\boldsymbol{V}|}\}$$

and the edge set

$$\boldsymbol{E} = \{E_1, E_2, \cdots, E_{|\boldsymbol{E}|}\}.$$

As introduced in Sect. 3.3.2, for a planar straight-line graph or more generally for a planar graph, besides the concepts vertex and edge, one more concept namely face is involved. So a generic planar straight-line graph $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{F}\}$ consists of the vertex set $\boldsymbol{V}$, the edge set $\boldsymbol{E}$, and the face set

$$\boldsymbol{F} = \{F_1, F_2, \cdots, F_{|\boldsymbol{F}|}\}.$$

For a planar straight-line graph $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{F}\}$, the PSLG's *raw graph*, denoted as $\boldsymbol{G}_R = \{\boldsymbol{V}, \boldsymbol{E}\}$, refers to the graph that stems from the PSLG by purely keeping the PSLG's inherent existence of being a graph but neglecting the PSLG's characteristic of being planar. The PSLG's *complete graph* refers to the graph $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{F}\}$ that contains complete information of the vertex set $\boldsymbol{V}$, the edge set $\boldsymbol{E}$, the face set $\boldsymbol{F}$, and specifies how the three sets are mutually adjoining. Without causing confusion, also call the the *complete graph* of the PSLG simply as the *graph* or just the PSLG itself.

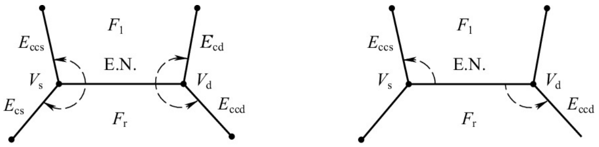
In practical applications, one may sometimes manually configure the complete graph of a planar straight-line graph. For example, one can manually index the vertices, the edges, and the faces of the PSLG and manually set their relationships. On the other hand, one may sometimes have a PSLG's raw graph initially and then needs to configure its complete graph automatically from the raw graph.

The *winged-edge representation* is a kind of typical representation for the planar straight-line graph [27], which stores for each edge the following augmented set of graph information (called *edge node*):

- V_s: the edge’s starting vertex.
- V_d: the edge’s destination vertex.
- F_l: the edge’s left face as viewed from V_s to V_d.
- F_r: the edge’s right face as viewed from V_s to V_d.
- E_cs: the clockwise edge successor around V_s.
- E_ccs: the counter-clockwise edge successor around V_s.
- E_cd: the clockwise edge successor around V_d.
- E_ccd: the counter-clockwise edge successor around V_d.

An example of such edge node is illustrated in Fig. 3.16a.

The *doubly-connected edge list representation* is another kind of typical representation for the planar straight-line graph [28], which simplifies the winged-edge representation by removing the elements of the clockwise edge successors E_cs and E_cd from the edge node structure, as illustrated in Fig. 3.16b.



(a) (b)

Fig. 3.16 Edge node **a** winged-edge representation; b doubly-connected edge list representation

3.3.4 Point-Location Problem

A fundamental problem related to planar straight-line graph applications is: Given a PSLG (treat it as a sub-division of the plane), how to determine a generic point on the plane is located in which face region of the PSLG? This is called the *point-location problem*.

An intuitive idea to solve the pointlocation problem is to traverse faces of the PSLG one by one and check whether the queried point is in certain traversed face polygon, yet such intuitive idea does not take advantage of special topology of the PSLG to benefit from algorithmic power.

The *slab method* [29] provides a way of taking advantage of special topology of the PSLG. More specifically, attach a horizontal line to each vertex of the PSLG and these horizontal lines divide the plane into $O(|\boldsymbol{V}|)$ slabs. Besides, each slab is divided by edges of the PSLG into at most $O(|\boldsymbol{E}|)$ sub-divisions that belong to pre-registered faces of the PSLG. According to analysis presented in Sect. 3.3.2, planar graphs are sparse and $|\boldsymbol{E}| = O(|\boldsymbol{V}|)$. So each slab is divided by edges of the PSLG into at most $O(|\boldsymbol{V}|)$ sub-divisions.

Locating the queried point in which slab involves $O(\log |\boldsymbol{V}|)$ computations in the spirit of binary searching and further locating the queried point in which sub-division of the found slab (i.e. finally locating the queried point in which face of the PSLG) also involves $O(\log |\boldsymbol{V}|)$ computations in the spirit of binary searching. So the slab method can solve the point-location problem via a total number of only $O(\log |\boldsymbol{V}|)$ computations.

The slab method requires a representation of $O(|\boldsymbol{V}|^2)$ complexity which increases rapidly as $|\boldsymbol{V}|$ increases. Such potentially-large representation complexity required by the slab method may be reduced considerably if faces of the PSLG can be roughly separated into distinct blocks especially regular blocks. If really so, one can first locate the queried point in its associated block and then apply the slab method locally within the block to further locate the queried point in its associated face of the PSLG.

3.4 Convex Geometry

Convex sets [30] are important geometric entities that play an indispensable role in computational geometry. A convex set $\boldsymbol{C}$ is a set such that for any pair of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$

$\in \boldsymbol{C}$ and any t satisfying $0 \leqslant t \leqslant 1$, one has

$$t\boldsymbol{P}_1 + (1-t)\boldsymbol{P}_2 \in \boldsymbol{C}, \tag{3.53}$$

or equivalently speaking, is a set such that for any group of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_k \in \boldsymbol{C}$ and any coefficients t_1, t_2, ..., t_k satisfying

$$t_1 \geqslant 0, \quad t_2 \geqslant 0, \quad \cdots \quad , \quad t_k \geqslant 0, \quad t_1 + t_2 + \cdots + t_k = 1, \tag{3.54}$$

one has

$$t_1\boldsymbol{P}_1 + t_2\boldsymbol{P}_2 + \cdots + t_k\boldsymbol{P}_k \in \boldsymbol{C}. \tag{3.55}$$

The linear combination of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_k$ in (3.55) with coefficients satisfying (3.54) is called their **convex combination**.

Given an arbitrary set $\boldsymbol{S}$ (not necessarily convex), the set of all convex combinations of points in $\boldsymbol{S}$ namely

$$\textbf{conv }\boldsymbol{S} \equiv \{t_1\boldsymbol{P}_1 + \cdots + t_k\boldsymbol{P}_k \mid \forall k, \boldsymbol{P}_i \in \boldsymbol{S}, t_i \geqslant 0, i = 1, \cdots, k; \sum_{i=1}^{k} t_i = 1\} \tag{3.56}$$

is called the **convex hull** of $\boldsymbol{S}$ [30, 31]. The convex hull **conv** $\boldsymbol{S}$ is the minimum convex set that contains $\boldsymbol{S}$.

If the positive coefficient requirement is removed from (3.54), one will have definition for an **affine set**, namely a set $\boldsymbol{A}$ such that for any group of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_k \in \boldsymbol{A}$ and any coefficients t_1, t_2, ..., t_k satisfying

$$t_1 + t_2 + \cdots + t_k = 1, \tag{3.57}$$

one has

$$t_1\boldsymbol{P}_1 + t_2\boldsymbol{P}_2 + \cdots + t_k\boldsymbol{P}_k \in \boldsymbol{A}. \tag{3.58}$$

The linear combination of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_k$ in (3.58) with coefficients satisfying (3.57) is called their **affine combination**. Similarly, given an arbitrary set $\boldsymbol{S}$, the set of all affine combinations of points in $\boldsymbol{S}$ namely

$$\textbf{aff }\boldsymbol{S} \equiv \left\{t_1\boldsymbol{P}_1 + \cdots + t_k\boldsymbol{P}_k \mid \forall k, \boldsymbol{P}_i \in \boldsymbol{S}, i = 1, \cdots, k; \sum_{i=1}^{k} t_i = 1\right\} \tag{3.59}$$

is called the **affine hull** of $\boldsymbol{S}$ [2], which is the minimum affine set that contains $\boldsymbol{S}$.

3.4.1 Convex Polygon

Convex polygons are a representative kind of practically tractable forms of convex sets. The convex hulls of a variety of geometric entities take the forms of convex polygons. For example, the convex hull of a planar straight-line graph is definitely a convex polygon, and the convex hull of a finite set of points on the plane is definitely a convex polygon also—In fact, the convex hull of a planar straight-line graph (PSLG) is identical to that of the finite set of PSLG vertices. Therefore, the problem of constructing the convex hull of a planar straight-line graph is essentially the same to the problem of constructing the convex hull of a finite set of planar points.

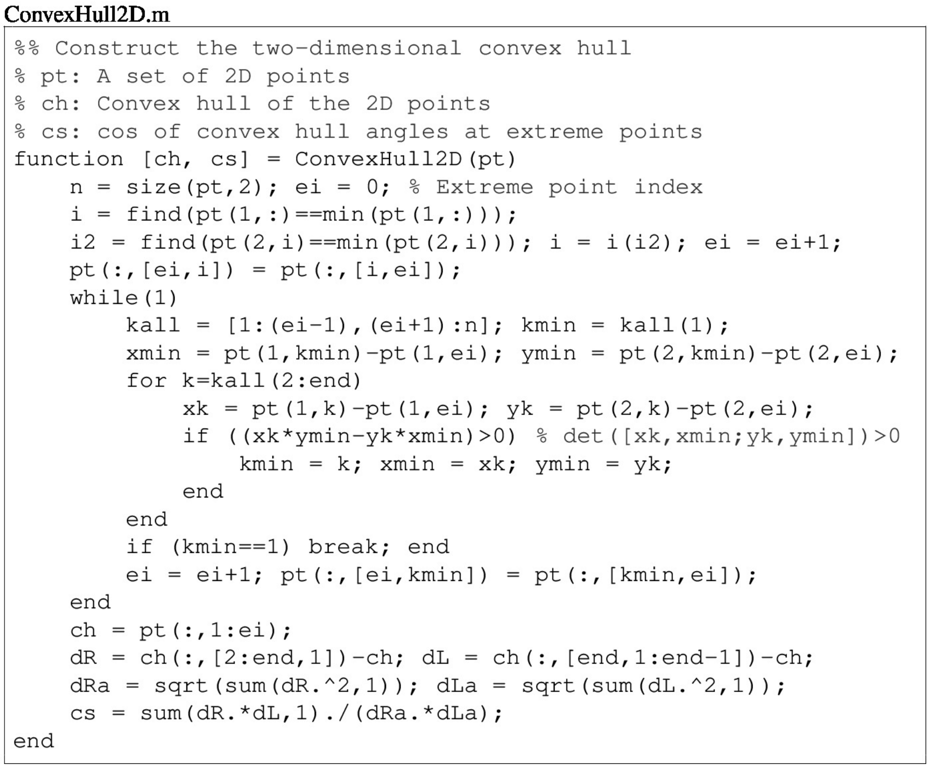
To construct the convex hull of a finite set of planar points, one may first find an *extreme point* [21] among them. This can be achieved by finding the point with the minimum horizontal coordinate (or the minimum vertical coordinate). If multiple points are with the minimum horizontal coordinate, then further find among them the one with the minimum vertical coordinate.

Upon finding an extreme point of the entire set of planar points, start from the extreme point, scan the other points from the perspective of the extreme point, and select among the other points the one with the minimum scanning angle, i.e. the one with the right-most scanning line. For the purpose, there is no real need to compute scanning angles associated with the other points via certain inverse trigonometric function such as arctan and then compare their scanning angles. Instead, one can take advantage of the following more efficient way. Given two points (x_1, y_1) and (x_2, y_2) (scanned from the perspective of the origin), to determine whether the scanning line of (x_1, y_1) is to the right of that of (x_2, y_2), just check whether the determinant

$$\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}$$

is positive or negative. Since the other points are scanned from the perspective of the extreme point, they must be located in the same half plane on one side of the extreme point [22] and will not encounter the problem of circular ambiguity during comparison of their scanning angles. If multiple points are with the minimum scanning angle, then select among them the furthest one from the extreme point.

Therefore, one can find a new extreme point, i.e. the second extreme point by scanning other points from the first extreme point. In a similar way, one can further find the third extreme point by scanning other points from the second extreme point, and continue to find the fourth extreme point, the fifth extreme point, and so on, until one arrives back again at the first extreme point. Then the convex hull of the finite set of planar points is finally constructed. The computational complexity of such scanning method for convex hull construction is $O(|\boldsymbol{V}|^2)$, where $|\boldsymbol{V}|$ denotes the number of the planar points. Matlab code for two-dimensional convex hull construction is given as follows.



ConvexHull2D.m

```
%% Construct the two-dimensional convex hull
% pt: A set of 2D points
% ch: Convex hull of the 2D points
% cs: cos of convex hull angles at extreme points
function [ch, cs] = ConvexHull2D(pt)
    n = size(pt,2); ei = 0; % Extreme point index
    i = find(pt(1,:)==min(pt(1,:)));
    i2 = find(pt(2,i)==min(pt(2,i))); i = i(i2); ei = ei+1;
    pt(:,[ei,i]) = pt(:,[i,ei]);
    while(1)
        kall = [1:(ei-1),(ei+1):n]; kmin = kall(1);
        xmin = pt(1,kmin)-pt(1,ei); ymin = pt(2,kmin)-pt(2,ei);
        for k=kall(2:end)
            xk = pt(1,k)-pt(1,ei); yk = pt(2,k)-pt(2,ei);
            if ((xk*ymin-yk*xmin)>0) % det([xk,xmin;yk,ymin])>0
                kmin = k; xmin = xk; ymin = yk;
            end
        end
        if (kmin==1) break; end
        ei = ei+1; pt(:,[ei,kmin]) = pt(:,[kmin,ei]);
    end
    ch = pt(:,1:ei);
    dR = ch(:,[2:end,1])-ch; dL = ch(:,[end,1:end-1])-ch;
    dRa = sqrt(sum(dR.^2,1)); dLa = sqrt(sum(dL.^2,1));
    cs = sum(dR.*dL,1)./(dRa.*dLa);
end
```

The convex hull of a finite set of planar points can be constructed in a computationally more efficient way, in the spirit of *divide-and-conquer*. The basic idea is: divide the original set into two sub-sets of planar points, and construct their convex hulls respectively. Then take advantage of the already ordered vertices of the two convex hulls (convex polygons) to efficiently merge the two convex hulls into one [2]. The computational complexity of such merging process can be just $O(|\boldsymbol{V}|)$ and hence the computational complexity of such convex hull construction method in the spirit of *divide-and-conquer* also satisfies the classical recursive relationship for algorithm computational complexity analysis, i.e.

$$C(|\boldsymbol{V}|) = 2\ C(|\boldsymbol{V}|/2) + O(|\boldsymbol{V}|),$$

which implies that the computational complexity of the convex hull construction method is

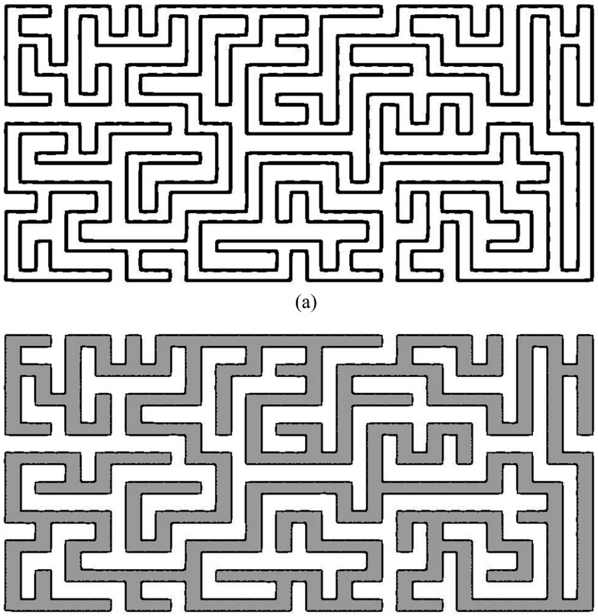
$$C(|\boldsymbol{V}|) = O(|\boldsymbol{V}|\ \log|\boldsymbol{V}|).$$

3.4.2 Point-Inclusion Problem

The *point-inclusion problem* can be regarded as a special (simplified) version of the *point-location problem* described in Sect. 3.3.4: Given a polygon (a special PSLG without interior edges and hence with a unique face enclosed by its edges), how can we determine whether a generic point on the plane is located inside the polygon or not?

At a glance, readers may think that one can still apply the slab method to solve the point-inclusion problem and may think as well that the point-inclusion problem would be solved much more efficiently than the point-location problem, because there is only a unique face enclosed by edges of the polygon or in other words there are only two faces under consideration: one inside the polygon and the other outside the polygon. Such small number of faces seems to be able to largely facilitate realization of the slab method.

It is true that one can still apply the slab method to solve the point-inclusion problem. However, the plausible convenience brought by the fact that there are only two faces under consideration is merely an illusion. Without any further a priori knowledge of the polygon, the slab method can solve the point-inclusion problem as efficiently as it solves the point-location problem for a generic PSLG , yet no more efficiently. To see the point, consider the maze-like polygon illustrated in Fig. 3.17. Figure 3.17a illustrates the original version of the maze-like polygon, whereas Fig. 3.17b illustrates an augmented version of the maze-like polygon with the inside face highlighted by light gray color.



(a)

(b)

Fig. 3.17 Maze-like polygon **a** original version; **b** augmented version with the inside face highlighted

Although there are only the inside face and the outside face under consideration, readers may reflect on a question: If one just takes a look at the original version of the maze-like polygon illustrated in Fig. 3.17a, can she or he easily tell which part of the image is inside the polygon and which part is outside the polygon? In fact, the question is indeed not easy, though not unsolvable. Moreover, if one further adds a number of line segments to divide the inside face of the maze-like polygon into multiple faces, readers would find that it seems no easier to determine whether a generic point is inside the original polygon or not than to locate it in which face of the divided polygon.

As presented in Sect. 3.3.4, the slab method can solve the point-location problem at a computational complexity of $O(\log|\boldsymbol{V}|)$ and a representation complexity of $O(|\boldsymbol{V}|^2)$, so if without any further a priori knowledge of the polygon, the slab method can solve the point-inclusion problem also at a computational complexity of $O(\log|\boldsymbol{V}|)$ and a representation complexity of $O(|\boldsymbol{V}|^2)$.

In contrast, the available *a priori* knowledge of convex polygons, namely *convexity*, can largely facilitate point-inclusion problem solving for a convex polygon. To determine whether a point P is inside a convex polygon $\boldsymbol{C}$, first select a point inside $\boldsymbol{C}$, say the gravity V_m of the vertices V_1, V_2, ..., $V_{|\boldsymbol{V}|}$ of $\boldsymbol{C}$, i.e.

$$V_\mathrm{m} = \frac{V_1+V_2+\cdots+V_{|\boldsymbol{V}|}}{|\boldsymbol{V}|}.$$

Second, connect lines from the gravity V_m to V_1, V_2, ..., $V_{|\boldsymbol{V}|}$ respectively to divide the plane into $|\boldsymbol{V}|$ polygon sectors

$$V_\mathrm{m}V_1V_2, \quad V_\mathrm{m}V_2V_3, \quad \cdots \quad , \quad V_\mathrm{m}V_{|\boldsymbol{V}|-1}V_{|\boldsymbol{V}|}, \quad V_\mathrm{m}V_{|\boldsymbol{V}|}V_1$$

and $|\boldsymbol{V}|$ associated plane sectors, as illustrated in Fig. 3.18.

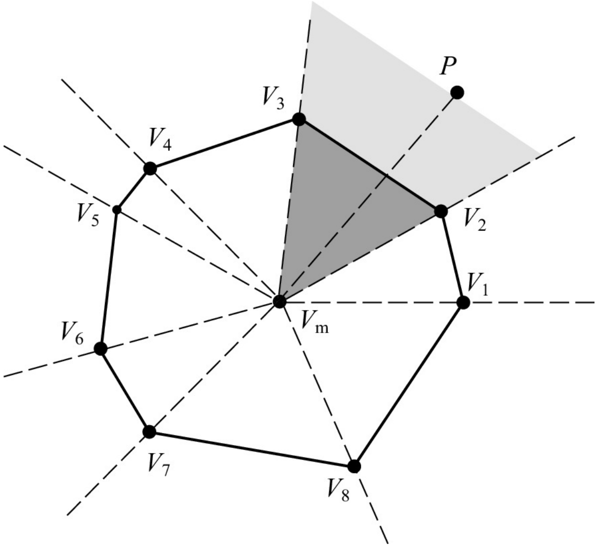


Fig. 3.18 Polygon-sector method for a convex polygon

Then locate the line $V_{\mathrm{m}}P$ in its corresponding plane sector via binary searching of its angle among angles of the lines that divide the plane into $|\boldsymbol{V}|$ sectors. As explained in Sect. 3.4.1, there is no real need to compute the line angles via certain inverse trigonometric function such as arctan; instead, one can resort to a more efficient way. Finally, determine whether the queried point P is inside its corresponding polygon sector (a triangle) using the method presented in Sect. 3.1.4. If so, one can conclude that the point P is inside the convex polygon $\boldsymbol{C}$; otherwise, not inside the convex polygon $\boldsymbol{C}$. This method of solving the point-inclusion problem for a convex polygon is called the *polygon-sector method*.

The polygon-sector method can solve the point-inclusion problem for a convex polygon also at a computational complexity of $O(\log|\boldsymbol{V}|)$, which seems to have no advantage over the slab method. However, the polygon-sector method achieves such $O(\log|\boldsymbol{V}|)$ computational efficiency at no extra representation cost, whereas the slab method has the same computational performance at a large extra representation complexity of $O(|\boldsymbol{V}|^2)$.

3.5 Closest Point Problems

In practical applications, one may encounter *closest point problems* [3, 32, 33], including how to find the point-pair with the closest distance among a set of points, and how to find the point among a given set of points closest to another given point. Closest point problems may be solved brutally by computing all involved distances and then traversing them to search for the minimum distance as well as the points associated with the minimum distance. Such brutal searching method is easily comprehensible but computationally undesirable. More efficient methods can be adopted in practice.

3.5.1 Find the Closest Pair Among Points

To find the point-pair with the closest distance among a set of points in a generic-dimensional space, it usually necessitates a computational complexity of $O(N^2)$, where N denotes the number of points under consideration. In contrast, for a special case of this general problem namely for the problem of finding the closest pair among a set of planar points, such computational complexity can be significantly reduced to $O(N\log N)$ via an elegant algorithm in the spirit of *divide-and-conquer*.

The key-point of the algorithm consists in dividing the set of points half by half for algorithmic recursion and exploiting a “clever” observation that the minimum distance associated with the entire set of points can only come from the minimum distance associated with one divided half of the set, the minimum distance associated with the other divided half of the set, and the minimum distance associated with a narrow band about the dividing line. Although the algorithm itself is of little importance for practical applications, yet its algorithmic elegance is worth recommendation. The author recommends [33] for readers to refer to details of the algorithm, though it was not [33] that proposed the algorithm.

3.5.2 Voronoi Diagram

For two generic planar points $\boldsymbol{P}_a$ and $\boldsymbol{P}_b$, define

$$D(\boldsymbol{P}_a, \boldsymbol{P}_b) \equiv \{\boldsymbol{P} \mid \|\boldsymbol{P} - \boldsymbol{P}_a\|_2 < \|\boldsymbol{P} - \boldsymbol{P}_b\|_2\} \tag{3.60}$$

and hence the *bisector* of $\boldsymbol{P}_a$ and $\boldsymbol{P}_b$, i.e.

$$B(\boldsymbol{P}_a, \boldsymbol{P}_b) \equiv \overline{D(\boldsymbol{P}_a, \boldsymbol{P}_b)} \cap \overline{D(\boldsymbol{P}_b, \boldsymbol{P}_a)} = \{\boldsymbol{P} \mid \|\boldsymbol{P} - \boldsymbol{P}_a\|_2 = \|\boldsymbol{P} - \boldsymbol{P}_b\|_2\} \tag{3.61}$$

separates the half-plane $D(\boldsymbol{P}_a, \boldsymbol{P}_b)$ containing $\boldsymbol{P}_a$ from the half-plane $D(\boldsymbol{P}_b, \boldsymbol{P}_a)$ containing $\boldsymbol{P}_b$. Here, the over-line notation denotes the *closure* of a set.[23]

Given a set of planar points

$$\boldsymbol{S} = \{\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_N\},$$

define the *Voronoi region* of a generic $\boldsymbol{P}$ with respect to $\boldsymbol{S}$ as

$$V_R(\boldsymbol{P}, \boldsymbol{S}) \equiv \cap_{\boldsymbol{P}' \in \boldsymbol{S}, \boldsymbol{P}' \neq \boldsymbol{P}} D(\boldsymbol{P}, \boldsymbol{P}') = \cap_{\boldsymbol{P}_i \neq \boldsymbol{P}, 1 \leqslant i \leqslant N} D(\boldsymbol{P}, \boldsymbol{P}_i). \tag{3.62}$$

Then the set defined as

$$V(\boldsymbol{S}) \equiv \cup_{\boldsymbol{P}_a, \boldsymbol{P}_b \in \boldsymbol{S}, \boldsymbol{P}_a \neq \boldsymbol{P}_b} \overline{V_R(\boldsymbol{P}_a, \boldsymbol{S})} \cap \overline{V_R(\boldsymbol{P}_b, \boldsymbol{S})} \tag{3.63}$$

is called the **Voronoi diagram** [34–36] of $\boldsymbol{S}$.

Intuitively speaking, the Voronoi diagram is a planar straight-line graph whose edges are called *Voronoi edges* and whose vertices are called *Voronoi vertices*. It divides the plane into N sub-divisions, i.e. Voronoi regions $V_R(\boldsymbol{P}_1, \boldsymbol{S})$, $V_R(\boldsymbol{P}_2, \boldsymbol{S})$, ..., $V_R(\boldsymbol{P}_N, \boldsymbol{S})$ associated with the N points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_N$ respectively. Each sub-division $V_R(\boldsymbol{P}_i, \boldsymbol{S})$ ($i \in \{1, \cdots, N\}$) consists of all points that are closer to $\boldsymbol{P}_i$ than to the other $N - 1$ points in the set.[24]

A Voronoi diagram example is illustrated in Fig. 3.19. The Voronoi diagram of a set of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_{10}$ is the planar straight-line graph with the vertex set

$$\boldsymbol{V} = \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{1\infty}, V_{2\infty}, V_{5\infty}, V_{8\infty}, V_{9\infty}, V_{11\infty}\}$$

and the edge set

$$\begin{aligned}\boldsymbol{E} = \{ \quad & V_1V_{1\infty}, V_1V_2, V_1V_3, V_2V_{2\infty}, V_2V_4, V_3V_8, V_3V_9, V_4V_5, V_4V_6, V_5V_{5\infty}, V_5V_{10}, \\ & V_6V_7, V_6V_{10}, V_7V_8, V_7V_{11}, V_8V_{8\infty}, V_8V_9, V_9V_{9\infty}, V_{10}V_{11}, V_{11}V_{11\infty}\},\end{aligned}$$

where $V_{1\infty}$, $V_{2\infty}$, $V_{5\infty}$, $V_{8\infty}$, $V_{9\infty}$, $V_{11\infty}$ denote imagined points along lines $V_1V_{1\infty}$, $V_2V_{2\infty}$, $V_5V_{5\infty}$, $V_8V_{8\infty}$, $V_9V_{9\infty}$, $V_{11}V_{11\infty}$ respectively and coincide at infinity. In other words, $V_{1\infty}$,

$V_{2\infty}$, $V_{5\infty}$, $V_{8\infty}$, $V_{9\infty}$, $V_{11\infty}$ can be treated as the same virtual vertex at infinity.

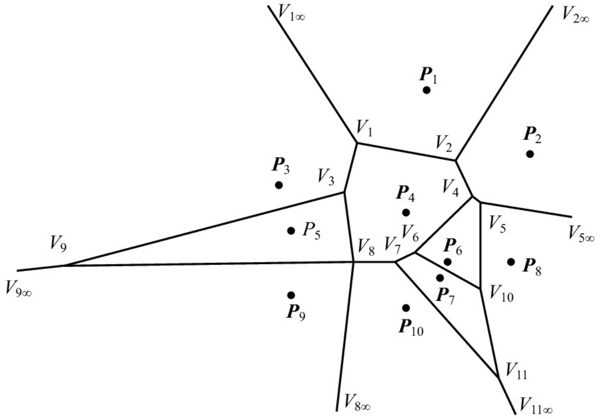


Fig. 3.19 Voronoi diagram of points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_{10}$

The Voronoi diagram of a N-point set can be efficiently constructed at a computational complexity of $O(N \log N)$ [3]. Besides, construction of Voronoi diagrams enable many closest point problems to be solved naturally and efficiently. For example, revisit the problem of finding the closest point-pair among a set of N points. One can first construct the Voronoi diagram of the set of points at a computational complexity of $O(N \log N)$. Then the closest point-pair can be found via a trivial traversal of the $O(N)$ number of Voronoi edges ,[25] and hence the total computational complexity of solving the problem of finding the closest point-pair among a set of N points is still $O(N \log N)$.

For another example, revisit the problem of finding the point among a given set of N points closest to a queried point. Again, one can first construct the Voronoi diagram of the set of points at a computational complexity of $O(N \log N)$. This computational cost is spent only once at the very beginning. Then for any queried point, finding its closest point in the set of N points is essentially the same to locating the queried point in its corresponding Voronoi region, i.e. its corresponding face of the Voronoi diagram (viewed as a planar straight-line graph), which can be solved efficiently at a computational complexity of $O(\log N)$ (see Sect. 3.3.4). If there are potentially a large amount of queried points, then it is worth constructing the Voronoi diagram first at the pre-processing cost of $O(N \log N)$ to significantly save time for processing the large amount of queried points latter.

References

1. B. Russell. *Principles of Mathematics* (Cambridge University Press, 1903)
2. F. Preparata, M. Shamos. *Computational Geometry: An Introduction* (Springer-Verlag, 1985)
3. J. Sack, J. Urrutia (eds.). *Handbook of Computational Geometry* (Elsevier Science, 2000)
4. J. Bonnans, J. Gilbert, C. Lemarechal, C. Sagastizabal. *Numerical Optimization: Theoretical and Practical Aspects* (Springer Science & Business Media, 2006)
5. D. Mitrinovic, P. Vasic, *Analytic Inequalities* (Springer-Verlag, Berlin Heidelberg, 1970) [Crossref][zbMATH]
6. N. Kwak, Principal component analysis by lp-norm maximization. IEEE Trans. Cybernet. **44**(5), 594–609 (2014) [Crossref]
7. J. Wang, Generalized 2-d principal component analysis by lp-norm for image analysis. IEEE Trans. Cybernet. **46**(3), 792–803 (2016) [Crossref]

8. I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in *International Conference on Learning Representations* (2015)
9. A. Kurakin, I. Goodfellow, S. Bengio. Adversarial examples in the physical world, in *International Conference on Learning Representations, Workshop Track* (2017)
10. A. Rosenfeld, J. Pfaltz, Sequential operations in digital picture processing. J. ACM **13**(4), 471–494 (1966) [Crossref][zbMATH]
11. P. Mahalanobis, On the generalized distance in statistics. Proc. Nat. Inst. Sci. India **2**(1), 49–55 (1936) [MathSciNet][zbMATH]
12. R.A. Horn, C.R. Johnson, *Matrix Analysis* (Cambridge University Press, 2012)
13. H. Koch, Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire. Arkiv for Matematik, Astronomi och Fysik **1**, 681–702 (1904) [zbMATH]
14. B. Mandelbrot, *Les Objets Fractals: Forme, Hasard et Dimension* (Flammarion, 1975)
15. B. Mandelbrot, *The Fractal Geometry of Nature* (Freeman, 1982)
16. T. Tao, *An Introduction to Measure Theory* (American Mathematical Society, 2011)
17. G. Cantor, Ueber unendliche, lineare punktmannichfaltigkeiten. Math. Annalen **21**, 545–591 (1883) [MathSciNet][Crossref]
18. F. Hausdorff, Dimension und äußeres maß. Mathematische Annalen **79**, 157–179 (1918)
19. H. Federer, *Geometric Measure Theory* (Springer-Verlag, 1996)
20. T. Tao, *An Epsilon of Room, I: Real Analysis* (American Mathematical Society, 2010)
21. K. Falconer, *Fractal Geometry: Mathematical Foundations and Applications* (John Wiley & Sons, 1990)
22. J. Gathen, J. Gerhard, *Modern Computer Algebra* (Cambridge University Press, 1999)
23. M. Farmer, *Application of Chaos and Fractals to Computer Vision* (Bentham Science Publishers, 2015)
24. L. Euler, Solutio problematis ad geometriam situs pertinentis. Commentarii Academiae Scientiarum Petropolitanae, **8**, 128–140 (1736)
25. I. Fáry, On straight line representation of planar graphs. Acta Scientiarum Math. **11**, 229–233 (1948)
26. D. West, *Introduction to Graph Theory* (Pearson Education, 2018)
27. B. Baumgart, A polyhedron representation for computer vision, in *National Computer Conference and Exposition* (1975), pp 589–596
28. D. Muller, F. Preparata, Finding the intersection of two convex polyhedra. Theoret. Comput. Sci. **7**(2), 217–236 (1978) [MathSciNet][Crossref][zbMATH]
29. D. Dobkin, R. Lipton, Multidimensional searching problems. SIAM J. Comput. **5**(2), 181–186 (1976) [MathSciNet][Crossref][zbMATH]
30. T. Bonnesen, W. Fenchel, *Theorie der konvexen Körper* (Julius Springer, 1934)
31. S. Boyd, L. Vandenberghe, *Convex Optimization* (Cambridge University Press, 2013)
32. M. Shamos, D. Hoey, Closest-point problems, in *Annual Symposium on Foundations of Computer Science* (1975), pp. 151 – 162
33. R. Sedgewick, *Algorithms* (Addison-Wesley Publishing Company, 1983)
34. G. Voronoï, Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. J. für die Reine und Angewandte Mathematik (Crelles Journal), **133**, 97–178 (1908)
35. G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloédres primitifs. J. für die Reine und Angewandte Mathematik (Crelles J.), **134**, 198–287 (1908)

36.
B. Delaunay, Sur la sphère vide. a la mémoire de georges voronoï. Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles, **6**, 793–800 (1934)

Footnotes

[1] The fairness of such imagination as well as the statement consists in the fact that they are logically realizable, be them actually realized by humans or not. The fairness of such kind contrasts the unfairness of imagination such as “a set of all elements that do not belong to oneself” or formalized as “$\boldsymbol{X} \equiv \{x \mid x \notin x\}$”, namely the *Russell paradox* [1], which is logically unrealizable.

[2] *Relativity theory* tells that space and time are essentially the same existence, just like the two sides of a coin, so it is indeed physically reasonable to measure the distance by time. However, when mentioning “essence of the so-called distance”, the author has no intention to arouse readers' sophisticated reflection on physics in the direction of relativity theory. The author intends that readers simply take the dialectic relationship between space and time as how they handle the two things in daily life.

[3] A typical example is the *geodesic* distance on the sphere-like earth surface.

[4] The degenerate case takes place when $\det\left([\boldsymbol{P}_1 \quad \boldsymbol{P}_2 \quad \boldsymbol{P}_3]\right) = 0$ or geometrically speaking, when the three vectors $\overrightarrow{\boldsymbol{OP}}_1$, $\overrightarrow{\boldsymbol{OP}}_2$, and $\overrightarrow{\boldsymbol{OP}}_3$ are located on the same plane, i.e. the origin $\boldsymbol{O}$ is located on the plane spanned by the three points.

[5] It is true that figurativeness of the term itself illustrated in Fig. 3.3 cannot be seen in all cities. It is indeed seen in cities such as Beijing, but cannot be seen in cities such as Paris which especially prefers triangular “quartiers”.

[6] In history, the concept of *Mahalanobis distance* was proposed in the context of statistics [11], and the relevant positive definite matrix $\boldsymbol{M}$ denoted the covariance of sample distribution originally in practice. On the other hand, applicability of the *Mahalanobis distance* is not limited to statistics but can be in general sense.

[7] This is like how people handle the concept of the ratio between a circle's circumference and its diameter. Although one can denote the concept as π abstractly, for practical handling (usage) of it, one has to indirectly understand π as the limit of a sequence of nested intervals

$$[3,4],\ [3.1,3.2],\ [3.14,3.15],\ [3.141,3.142],\ [3.1415,3.1416],\\ [3.14159,3.14160],\ [3.141592,3.141593]\ \cdots$$

or the limit of a sequence of other forms such as continued fractions

$$3,\quad 3+\tfrac{1}{7}=\tfrac{22}{7},\quad 3+\frac{1}{7+\dfrac{1}{15}}=\tfrac{333}{106},\quad 3+\frac{1}{7+\dfrac{1}{15+\dfrac{1}{1}}}=\tfrac{355}{113},\quad \ldots$$

[8] Why for the convergence? Readers may resort to the *Cauchy criterion* to verify this conclusion. Simply speaking, given an arbitrarily infinitesimal ϵ, one can always find certain N such that for any $j,k > N$ the maximum distance between $\boldsymbol{S}_j$ and $\boldsymbol{S}_k$ is within ϵ.

[9] See explanations given in the footnote at the very beginning of Chap. 3.

[10] Originally in [17], *Cantor* did not directly propose his set visually as illustrated in Fig. 3.11, but proposed it indirectly via a 3-adic fraction number (i.e. base-three positional fraction number) representation

$$z = \frac{c_1}{3} + \frac{c_2}{3^2} + \cdots + \frac{c_v}{3^v} + \cdots$$

where each $c_v \in \{0, 2\}$, namely each c_v in the 3-adic representation takes a value of 0 or 2 but not 1.

11 More specifically, the *Hausdorff measure* refers to the Hausdorff “outer (exterior) measure” conveyed by its original name “äußere maß”. Some extra words hover over acceptable omission of “outer” in mathematical conventions: There is clear distinction between “outer” and “inner” in definition of the *Jordan measure* that resorts to *elementary sets* (i.e. finite unions of boxes). Since following measures such as the famous *Lebesgue measure* that supersede the Jordan measure, explicit conceptualization of “inner” in them seems to bring no gain beyond the Jordan measure. Besides, for actual usage of the measure, “outer” seems to bear a sense more practical than “inner”. It is like when one counts the number of audiences that a cinema can accommodate, one will not only consider the place occupied just by the body of each audience (somewhat analogous to the “inner” measure), but will consider the place occupied by the audience and the seat together (somewhat analogous to the “outer” measure).

12 Lower-dimensional geometric entities, even the entire space of a lower dimension, have the zero measure in higher-dimensional space. For example, even the entire two-dimensional plane still has the zero measure from the perspective of the three-dimensional space.

13 Note that no matter how part of the stochastic snow fractal is generated iteratively from a line segment, the part is definitely constrained within a parallelogram (more concretely a rhombus) covered also by the disk that covers the line segment.

14 The Γ shape fractal is a kind of fractal modelling for computational complexity of the fast multi-digit number multiplication algorithm proposed by *Karatsuba* [22], as briefly reviewed in the footnote in Sect. 1.5 in Chap. 1.

15 In fact, graphs are effective not only for topological modelling of geometric entities, but also for more general “relationship” modelling of a much wider range of entities such as societies. Here, the author focuses on geometric entities.

16 In the adjacency matrix representation example given here, numbers mean edge numbers among vertices. For example, the number 2 associated with *A* and *B* means there are two edges between *A* and *B*. It is worth noting that numbers in an adjacency matrix representation may also mean distances among vertices sometimes. What if for a graph one needs to represent multiple edges of different lengths between two vertices? Readers may reflect whether it is common to encounter such situation and how to flexibly handle it in case.

17 A generic *n*-by-*n* square matrix $\boldsymbol{M} = [m_{ij}]$ is *diagonally dominant* [12] if

$$|m_{ii}| \geqslant \sum_{j \neq i} |m_{ij}| \qquad \text{for all } i = 1, \cdots, n$$

and is *strictly diagonally dominant* if

$$|m_{ii}| > \sum_{j \neq i} |m_{ij}| \qquad \text{for all } i = 1, \cdots, n.$$

18 The *Gersgorin theorem* [12] states that given a generic *n*-by-*n* square matrix $\boldsymbol{M} = [m_{ij}]$, consider the *deleted absolute row sums* of $\boldsymbol{M}$

$$R_i(\boldsymbol{M}) \equiv \sum_{j \neq i} |m_{ij}| \qquad \text{for all } i = 1, \cdots, n$$

and consider the *n Gersgorin discs*

$$\{z \in \mathrm{C} \mid \|z - m_{ii}\| \leqslant R_i(\boldsymbol{M})\} \qquad \text{for all } i = 1, \cdots, n.$$

Eigenvalues of $\boldsymbol{M}$ are in the union of Gersgorin discs

$$G(\boldsymbol{M}) \equiv \cup_{i=1}^{n}\{z \in \mathrm{C} \mid \|z-m_{ii}\| \leqslant R_i(\boldsymbol{M})\}.$$

[19] Graphs have different ways of being embedded into the plane. Some graphs can have ways of being embedded into planar graphs; for example, a tetrahedron-like graph can be embedded into a planar graph. In contrast, other graphs cannot be embedded into planar graphs, no matter how they are embedded into the plane. Therefore, in somehow strict sense, whether a graph is indeed planar or not may be defined by whether it has certain way of being embedded into a planar graph or it has no way of being embedded so. However, the author does not scrutinize the "planar" definition in such strict sense, or in other words, the author does not scrutinize any graph's potential of being embeddable into a planar graph or not. Instead, the author simply treats different results of graph embedding on the plane as different graphs, regardless of whether they are potentially homeomorphic in three-dimensional space or topologically identical in other senses.

[20] If a planar graph has infinite-length edges, then all the infinite-length edges are imagined to coincide at a virtual vertex at infinity. $V_4 = \infty$ is such virtual vertex at infinity for the planar graph example illustrated in Fig. 3.15b.

[21] For a set whose convex hull is a convex polygon, an extreme point of it can be simply understood as a vertex of its convex hull. For a generic set, an extreme point of it is a point of it such that it does not contain any pair of two other points whose spanned line segment contains the point within.

[22] The two sides of the extreme point are not necessarily regular, i.e. vertical or horizontal, but can be inclined.

[23] The over-line notation plays a role in both topology and set theory. It conventionally denotes the closure of a set in the former, whereas it denotes the complement of a set in the latter.

[24] The concept "Voronoi diagram" is named after *G. Voronoï* for his study of generic-dimensional polyhedrons and "paralléloèdres" (convex polyhedrons that can pave the corresponding generic-dimensional space via operations of translation groups) from the perspective of quadratic forms [34, 35]. However, for the Voronoi diagram, neither the term itself nor its usual meaning interpreted nowadays, was described directly by Voronoï himself in these original articles of his own. It was *B. Delaunay* who, in his works on partitioning generic-dimensional space via corresponding generic-dimensional tetrahedrons (usually known as *Delaunay triangulation* nowadays) [36], borrowed important concepts such as "sommet primitif" (in Delaunay's paraphrased term) namely "sommet simple" (in Voronoï's original term) from Voronoï's previous works and made "Voronoi" debut in terms of bearing its usual meaning interpreted nowadays. Such "sommets primitifs" mentioned by Delaunay in [36] actually refer to *Voronoi vertices* in the Voronoi diagram.

[25] Note that the Voronoi diagram is a planar graph and is sparse.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_4

4. Projective Geometry and Camera Model

Hao Li[1]
(1) Shanghai Jiao Tong University, Shanghai, China

4.1 Projective Geometry
4.2 Camera Model
4.2.1 Pinhole Camera Model
4.2.2 Extrinsic and Intrinsic Parameters
4.2.3 Perspective Mapping
4.2.4 Revisit Common Sense of Projective Geometry
4.3 Homography
4.3.1 Bijective Mapping Between Two-Dimensional Planes
4.3.2 Homography Calibration
4.3.3 Inverse Perspective Mapping
4.4 Image Distortion
4.4.1 Distortion-Considered Camera Model
4.4.2 Inverse Distortion Model and Distortion Rectification
4.5 Stereo Vision
4.5.1 Why Stereo Vision Is “Natural”?
4.5.2 Epipolar Constraint: Fundamental and Essential Matrices
4.5.3 Rectified Stereo Vision
4.5.4 Binocular Disparity
References

4.1 Projective Geometry

Projective geometry has long since been researched and humans’ knowledge of it has long since been applied. A vivid example of applying knowledge of projective geometry in arts is illustrated in Fig. 1.9.

To further appreciate beauty of projective geometry, one may take a look at several famous theorems of geometry, namely the *Pappus theorem*, the *Desargues theorem*, and the *Pascal theorem* [1, 2].

As illustrated in Fig. 4.1a, given a line A–B–C with three consecutive points A, B, and C on it, and given another line D–E–F with three consecutive points D, E, and F on this second line. Connect the points A, B, and C to the points D, E, and F. Suppose the line A–E and the line B–D intersect at a point L, the line A–F and the line C–D intersect at a point M, the line B–F and the line C–E intersect at a point N. Then the *Pappus theorem* states that *the three intersection points* L, M, *and* N *must be located on the same line*.

Proof Suppose the line A–E and the line B–F intersect at a point X, the line C–D intersects the line A–E and the line B–F at a point Y and a point Z respectively, as illustrated in Fig. 4.1b. Apply the Menelaus theorem (see Sect. 3.2.4 in Chap. 3) for the triangle X–Y–Z with following triples of colinear points respectively, namely $\{A, M, F\}$, $\{B$

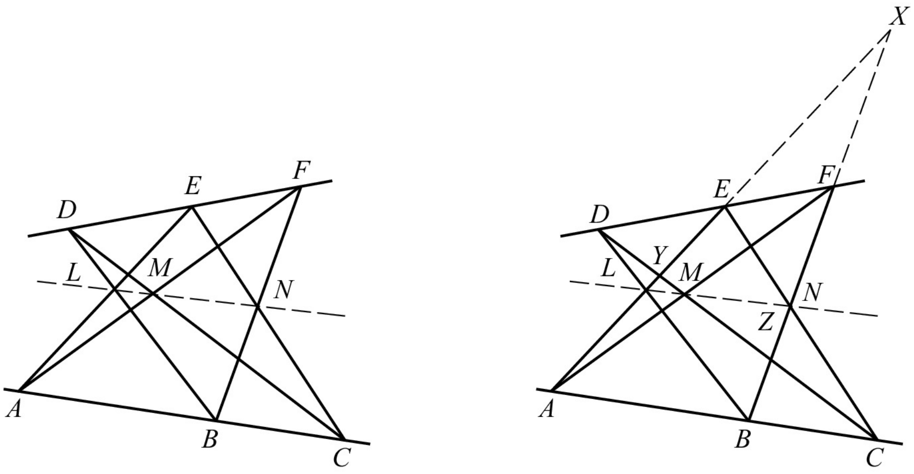
$, L, D\}$, $\{C, N, E\}$, $\{A, B, C\}$, and $\{D, E, F\}$. Then there are a number of equalities:

$$\begin{aligned}
\frac{|ZM|}{|MY|} \cdot \frac{|YA|}{|AX|} \cdot \frac{|XF|}{|FZ|} &= 1, \\
\frac{|YL|}{|LX|} \cdot \frac{|XB|}{|BZ|} \cdot \frac{|ZD|}{|DY|} &= 1, \\
\frac{|XN|}{|NZ|} \cdot \frac{|ZC|}{|CY|} \cdot \frac{|YE|}{|EX|} &= 1, \\
\frac{|YC|}{|CZ|} \cdot \frac{|ZB|}{|BX|} \cdot \frac{|XA|}{|AY|} &= 1, \\
\frac{|XE|}{|EY|} \cdot \frac{|YD|}{|DZ|} \cdot \frac{|ZF|}{|FX|} &= 1.
\end{aligned}$$

Multiply the left sides of above equalities and obtain

$$\begin{aligned}
\varPi_{\text{left}} = \;& \frac{|ZM|}{|MY|} \cdot \frac{|YA|}{|AX|} \cdot \frac{|XF|}{|FZ|} \cdot \frac{|YL|}{|LX|} \cdot \frac{|XB|}{|BZ|} \cdot \frac{|ZD|}{|DY|} \cdot \frac{|XN|}{|NZ|} \cdot \frac{|ZC|}{|CY|} \cdot \frac{|YE|}{|EX|} \cdot \\
& \frac{|YC|}{|CZ|} \cdot \frac{|ZB|}{|BX|} \cdot \frac{|XA|}{|AY|} \cdot \frac{|XE|}{|EY|} \cdot \frac{|YD|}{|DZ|} \cdot \frac{|ZF|}{|FX|} \\
= \;& \frac{|ZM|}{|MY|} \cdot \frac{|YL|}{|LX|} \cdot \frac{|XN|}{|NZ|} = 1.
\end{aligned}$$

Apply the (inverse) Menelaus theorem for the triangle X–Y–Z with the triple of points $\{L, M, N\}$ to know that the three intersection points L, M, and N are located on the same line. The proof is done.□



(a) (b)

Fig. 4.1 Pappus theorem **a** original configuration; **b** configuration with auxiliary lines

Above proof of the Pappus theorem necessitates some mathematical skills that are not so easy to figure out at a glance. In contrast, the author next gives a much easier proof by taking advantage of projective geometry, more specifically, by taking advantage of several aspects of invariance in projective geometry:

- *Invariance 1*: A point after perspective mapping (or perspective transform) is still a point.
- *Invariance 2*: A line after perspective mapping is still a line.
- *Invariance 3*: Coinciding lines after perspective mapping still coincide.
- *Invariance 4*: Colinear points after perspective mapping are still colinear.

Above aspects of invariance are intuitive, yet a rigorous algebraic proof of them can also be made, which will be postponed after presentation of the camera model.

Proof Imagine the original configuration illustrated in Fig. 4.1 as the projected image of a real-world scenario, where the first group of parallel tracks $\{A\text{–}D, B\text{–}D, C\text{–}D\}$ lead to infinity at the vanishing point *D*, the second group of parallel tracks $\{A\text{–}E, B\text{–}E, C\text{–}E\}$ lead to infinity at the vanishing point *E*, and the third set of parallel tracks $\{A\text{–}F, B\text{–}F, C\text{–}F\}$ lead to infinity at the vanishing point *F*,[1] as illustrated in Fig. 4.2.

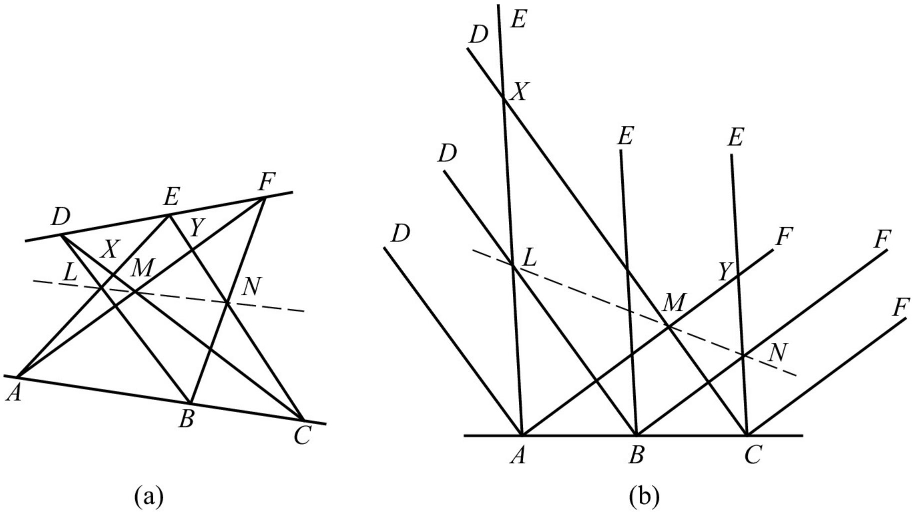


Fig. 4.2 Pappus theorem **a** original configuration imagined as projected image; **b** imagined real-world scenario

As parallel lines keep ratios of line segments cut by them, one has

$$\frac{|XL|}{|LA|} = \frac{|CB|}{|BA|} = \frac{|CN|}{|NY|}$$

in the imagined real-world scenario, which implies that *L*, *M*, and *N* are colinear. According to the *Invariance 4*, the three intersection points *L*, *M*, and *N* are also colinear in the original configuration. The proof is done.□

One can see how projective geometry brings grace and conciseness to the proof of the Pappus theorem. It can also bring grace and conciseness to the proof of the Desargues theorem.

As illustrated in Fig. 4.3, given a triangle *A*–*B*–*C* and another triangle *D*–*E*–*F*, suppose the line *A*–*D*, the line *B*–*E*, and the line *C*–*F* coincide at a point *O*; the line *C*–*A* and the line *F*–*D* intersect at a point *L*; the line *A*–*B* and the line *D*–*E* intersect at a point *M*; the line *B*–*C* and the line *E*–*F* intersect at a point *N*. Then the *Desargues theorem* states that *the three intersection points L*, *M*, *and N must be located on the same line*.

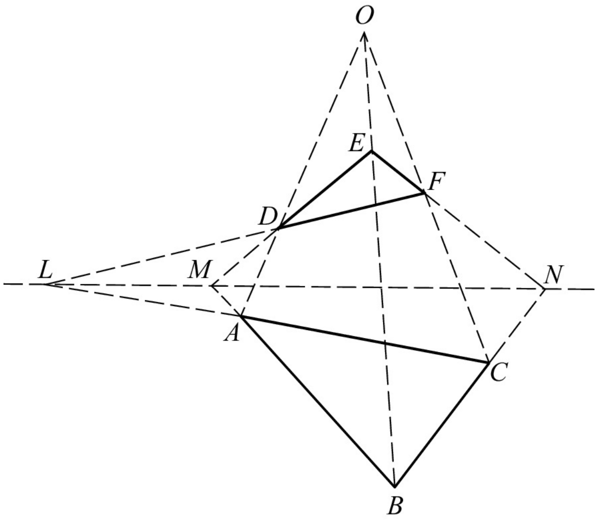


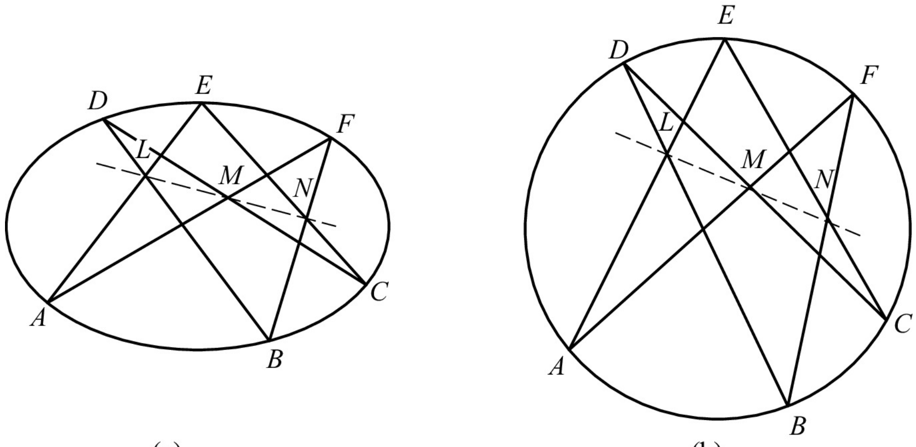
Fig. 4.3 Desargues theorem: configuration imagined as projected image of a tetrahedron intersected by two planes

One can also resort to the Menelaus theorem to provide a proof of the Desargues theorem, yet details are omitted here. Instead, the author directly demonstrates how to take advantage of projective geometry to figure out a convenient proof of the Desargues theorem.

Proof Imagine the original planar configuration illustrated in Fig. 4.3 as the projected image of a tetrahedron intersected by two planes in the three-dimensional space. The tetrahedron is formed by the four triangles *O*–*A*–*B*, *O*–*B*–*C*, *O*–*C*–*A*, and *A*–*B*–*C*. The two planes are spanned by the triangle *A*–*B*–*C* and the triangle *D*–*E*–*F* respectively.

L is the intersection point of two lines, i.e. the line *C*–*A* and the line *F*–*D* that are located on the two planes respectively, so it must be located on the intersection line of the two planes. Similarly, *M* and *L* must also be located on the intersection line of the two planes. Therefore, the three intersection points *L*, *M*, and *N* are located on the same line. The proof is done.□

The beauty of projective geometry can especially be demonstrated in the proof of the *Pascal theorem*. As illustrated in Fig. 4.4a, given an ellipse *A*–*B*–*C*–*F*–*E*–*D*, connect the points *A*, *B*, and *C* to the points *D*, *E*, and *F*. Suppose the line *A*–*E* and the line *B*–*D* intersect at a point *L*, the line *A*–*F* and the line *C*–*D* intersect at a point *M*, the line *B*–*F* and the line *C*–*E* intersect at a point *N*. Then the *Pascal theorem* states that *the three intersection points L, M, and N must be located on the same line*.



(a) (b)

Fig. 4.4 Pascal theorem **a** original ellipse configuration imagined as projected image; **b** imagined circle counterpart in the real-world

Before continuing, readers may take a while to reflect how to prove the Pascal theorem directly with the ellipse configuration, either in pure geometric way or in analytical geometric way. This is indeed not so easy. On the other hand, projective geometry can largely facilitate the proof of the Pascal theorem.

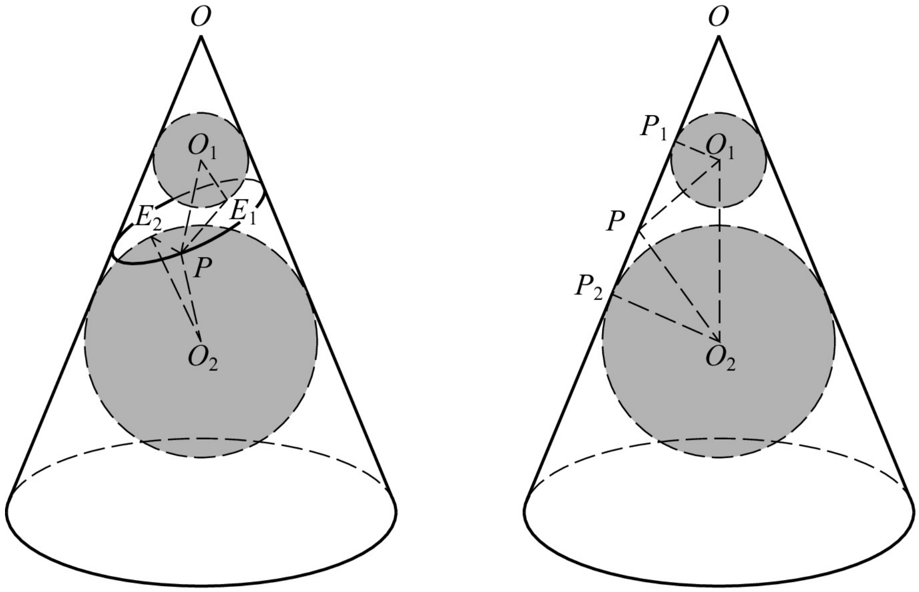
Proof Both the circle and the ellipse are *conic curves* or *conic sections* , their bijective relationship can be demonstrated by Fig. 4.5. The cone (denoted by its summit *O*) has a generic horizontal circle section, the size of which can be adjusted by changing the section height in the cone. Given a generic closed section $\big($denoted by the section of the plane spanned by $\{E_1, E_2, P\}\big)$, *it must be an ellipse*.

To know this, construct two inscribed spheres (denoted by their centers O_1 and O_2 respectively) of the cone *O*. They are also inscribed to the upper and lower sides of the inclined conic section respectively. O_1–E_1 and O_2–E_2 are perpendicular to the inclined conic section. The point P denotes a generic point on the inclined conic curve . Besides, consider the vertical section O–O_1–O_2–P_2–P–P_1 of the cone, where O_1–P_1 and O_2–P_2 are perpendicular to the line O–P, as illustrated in Fig. 4.5b. Then

$$\begin{aligned} |E_1P| &= \sqrt{|O_1P|^2 - |O_1E_1|^2} = \sqrt{|O_1P|^2 - |O_1P_1|^2} = |P_1P|, \\ |E_2P| &= \sqrt{|O_2P|^2 - |O_2E_2|^2} = \sqrt{|O_2P|^2 - |O_2P_2|^2} = |P_2P|, \end{aligned}$$

so

$$|E_1P|+|E_2P| = |P_1P|+|P_2P| = |P_1P_2| \quad (\text{a constant}),$$



(a) (b)

Fig. 4.5 Conic curves: bijective mapping between the ellipse and the circle

which implies that the inclined conic curve is an ellipse. The shape of the ellipse can be adjusted by varying the slope of the cone and by varying the section pose.

As the bijective relationship between the circle and the ellipse has been clarified, then imagine the original ellipse configuration as the projected image of a circle counterpart in the real-world, as illustrated in Fig. 4.4b. According to the four aspects of invariance in projective geometry, one only needs to prove the Pascal theorem for the circle configuration, which equivalently implies validity of the Pascal theorem for any generic ellipse configuration.

Suppose the line *A*–*E* and the line *B*–*F* intersect at a point *X*, the line *C*–*D* intersects the line *A*–*E* and the line *B*–*F* at a point *Y* and a point *Z* respectively, as illustrated in Fig. 4.6. Apply the Menelaus theorem

for the triangle *X*–*Y*–*Z* with following triples of colinear points respectively, namely $\{A, M, F\}$, $\{B, L, D\}$, and $\{C, N, E\}$. Then there are a number of equalities:

$$
\begin{aligned}
\frac{|ZM|}{|MY|} \cdot \frac{|YA|}{|AX|} \cdot \frac{|XF|}{|FZ|} &= 1, \\
\frac{|YL|}{|LX|} \cdot \frac{|XB|}{|BZ|} \cdot \frac{|ZD|}{|DY|} &= 1, \\
\frac{|XN|}{|NZ|} \cdot \frac{|ZC|}{|CY|} \cdot \frac{|YE|}{|EX|} &= 1.
\end{aligned}
$$

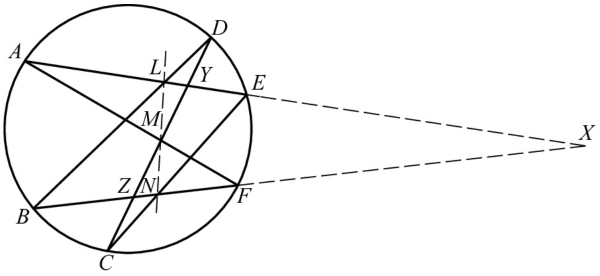


Fig. 4.6 Pascal theorem: circle configuration with auxiliary lines

Multiply the left sides of above equalities and obtain

$$\begin{aligned}\mathit{\Pi}_{\text{left}} &= \frac{|ZM|}{|MY|}\cdot\frac{|YA|}{|AX|}\cdot\frac{|XF|}{|FZ|}\cdot\frac{|YL|}{|LX|}\cdot\frac{|XB|}{|BZ|}\cdot\frac{|ZD|}{|DY|}\cdot\frac{|XN|}{|NZ|}\cdot\frac{|ZC|}{|CY|}\cdot\frac{|YE|}{|EX|}\\ &= \frac{|ZM|}{|MY|}\cdot\frac{|YL|}{|LX|}\cdot\frac{|XN|}{|NZ|}\cdot\frac{|XF|\cdot|XB|}{|AX|\cdot|EX|}\cdot\frac{|YA|\cdot|YE|}{|CY|\cdot|DY|}\cdot\frac{|ZD|\cdot|ZC|}{|BZ|\cdot|FZ|}\\ &= \frac{|ZM|}{|MY|}\cdot\frac{|YL|}{|LX|}\cdot\frac{|XN|}{|NZ|} = 1,\end{aligned}$$

where three equalities about line sections of the circle are used, namely

$$|XF|\cdot|XB| = |AX|\cdot|EX|,\quad |YA|\cdot|YE| = |CY|\cdot|DY|,\\ |ZD|\cdot|ZC| = |BZ|\cdot|FZ|.$$

Apply the (inverse) Menelaus theorem for the triangle X–Y–Z with the triple of points $\{L, M, N\}$ to know that the three intersection points L, M, and N are located on the same line. The proof is done.□

The author has demonstrated beauty as well as utility of projective geometry via vivid proofs of three famous theorems of geometry, namely the *Pappus theorem*, the *Desargues theorem*, and the *Pascal theorem*. In next sub-section, the author will present algebraic foundations of projective geometry in the context of computer vision, which are indispensable in a wide range of practical applications.

4.2 Camera Model

In the context of computer vision, the **camera model** formalizes algebraic foundations of projective geometry.

4.2.1 Pinhole Camera Model

If distortion is neglected, then how a three-dimensional (3D) world point is projected onto the image of a computer vision system can be described by the **pinhole camera model** , as illustrated in Fig. 4.7.

There are three important coordinates systems in the model, namely the **world coordinates system**, the **camera coordinates system**, and the **image coordinates system**.[2] The image coordinates system has already been introduced in Sect. 2.1 in Chap. 2.

In practical applications, the world coordinates system $O_{\text{w}}X_{\text{w}}Y_{\text{w}}Z_{\text{w}}$ is attached to an appropriate entity in the three-dimensional real-world and is used to describe the position of each 3D world point with respect to the entity. How to establish the world coordinates system depends on concrete applications. For example, it can be attached to the ground and wall of a classroom for the surveillance camera in the classroom; it can be established on the body of an intelligent vehicle for the on-vehicle camera.

The camera coordinates system $O_{\text{c}}X_{\text{c}}Y_{\text{c}}Z_{\text{c}}$ is always attached to the camera and is used to describe the position of each 3D world point with respect to the camera. By convention, the axis $O_{\text{c}} - Z_{\text{c}}$ is chosen to be along the camera *principal axis* towards the camera view direction, and the axes $O_{\text{c}} - X_{\text{c}}$ and $O_{\text{c}} - Y_{\text{c}}$ are chosen to be “virtually parallel” [3] with the axes $O_{\text{I}} - U$ and $O_{\text{I}} - V$ of the image coordinates system. The principal axis intersects the plane of the image coordinates system at a point called the *principal point* .

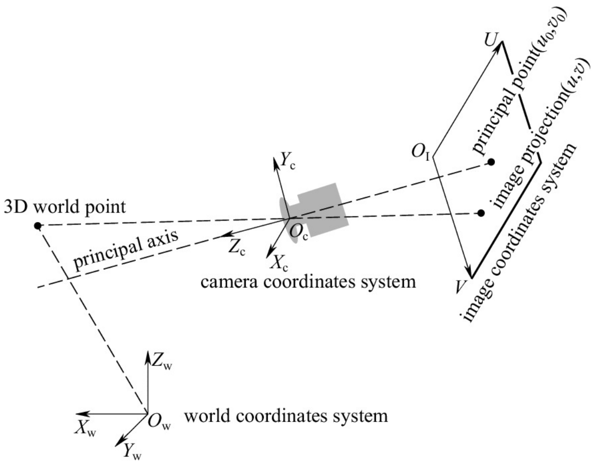

U
principal point(u_0,v_0)
image projection(u,v)
O_I
image coordinates system
V
Y_c
3D world point
O_c
principal axis
Z_c
X_c
camera coordinates system
Z_w
X_w
O_w
world coordinates system
Y_w

Fig. 4.7 Pinhole camera model

The camera model formalizes the relationships or transforms among the world coordinates system, the camera coordinates system, and the image coordinates system, such that based on the camera model, one can derive the image projection of a given 3D world point on one hand, and derive partially the 3D world counterpart of a given image point on the other hand.

4.2.2 Extrinsic and Intrinsic Parameters

The transform between the world coordinates system and the camera coordinates system consists of a rotation and a translation, formalized as

$$\begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix} = \boldsymbol{R}_\mathrm{wc} \begin{bmatrix} x_\mathrm{w} \\ y_\mathrm{w} \\ z_\mathrm{w} \end{bmatrix} + \boldsymbol{T}_\mathrm{wc} \tag{4.1}$$

or compactly as

$$\boldsymbol{P}_\mathrm{c} = \boldsymbol{R}_\mathrm{wc} \boldsymbol{P}_\mathrm{w} + \boldsymbol{T}_\mathrm{wc},$$

where $\boldsymbol{P}_\mathrm{w} \equiv \begin{bmatrix} x_\mathrm{w} & y_\mathrm{w} & z_\mathrm{w} \end{bmatrix}^\mathrm{T}$ and $\boldsymbol{P}_\mathrm{c} \equiv \begin{bmatrix} x_\mathrm{c} & y_\mathrm{c} & z_\mathrm{c} \end{bmatrix}^\mathrm{T}$ denote the coordinates of a generic 3D world point $\boldsymbol{P}$ expressed in the world coordinates system and in the camera coordinates system respectively.

The transform between the camera coordinates system and the image coordinates system is formalized as

$$u = \alpha \tfrac{x_\mathrm{c}}{z_\mathrm{c}} + \gamma \tfrac{y_\mathrm{c}}{z_\mathrm{c}} + u_0, \tag{4.2a}$$

$$v = \beta \tfrac{y_\mathrm{c}}{z_\mathrm{c}} + v_0. \tag{4.2b}$$

If the photosensitive plane could be ideally manufactured and aligned in the camera, then the transform between the camera coordinates system and the image coordinates system would be formalized as

$$u = \alpha \tfrac{x_\mathrm{c}}{z_\mathrm{c}} + u_0, \tag{4.3a}$$

$$v \ = \alpha \tfrac{y_\mathrm{c}}{z_\mathrm{c}} + v_0. \tag{4.3b}$$

As the discrepancy between the parameters α and β is usually small and the parameter γ is small as well in practical applications, approximating (4.2) by the simpler ideal form (4.3) may be an acceptable practice sometimes.

Formalize (4.2) equivalently as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix} \tag{4.4}$$

or compactly as

$$\lambda \boldsymbol{P}_\mathrm{i}^A = \boldsymbol{A}\boldsymbol{P}_\mathrm{c} \quad \left(\boldsymbol{A} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \right),$$

where $\boldsymbol{P}_\mathrm{i}^A \equiv \begin{bmatrix} u & v & 1 \end{bmatrix}^\mathrm{T}$ denotes the augmented vector of the image point $\boldsymbol{P}_\mathrm{i} \equiv \begin{bmatrix} u & v \end{bmatrix}^\mathrm{T}$.

Note that the scaling factor λ in (4.4) is actually the same to z_c, based on which the equivalence between (4.4) and (4.2) can be easily verified. Readers may wonder why not directly put z_c in place of λ in (4.4). In fact, one can put z_c in place of λ, yet (4.4) which uses an extra variable λ is preferred in practical applications, especially for convenience of theoretical analysis. The reason is to highlight the fact that different triplets of $(x_\mathrm{c}, y_\mathrm{c}, z_\mathrm{c})$ are actually equivalent in determining the image projection $\boldsymbol{P}_\mathrm{i}$, if they differ from each other only by a scaling factor λ.

Besides, the form of (4.4) facilitates inverse inference from the image coordinates $\boldsymbol{P}_\mathrm{i}$ to the camera coordinates $\boldsymbol{P}_\mathrm{c}$ (viewed as a *projective line*):

$$\begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix} = \lambda \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.5}$$

or compactly as

$$\boldsymbol{P}_\mathrm{c} = \lambda \boldsymbol{A}^{-1} \boldsymbol{P}_\mathrm{i}^A.$$

Otherwise, if z_c is put in place of λ, the form of (4.5) would become somewhat tricky and unnatural.

The parameters of the rotation matrix $\boldsymbol{R}_\mathrm{wc}$ and the translation vector $\boldsymbol{T}_\mathrm{wc}$ in (4.1) are called *camera extrinsic parameters*, whereas the parameters α, β, γ, u_0, and v_0 in (4.2) and (4.4) are called *camera intrinsic parameters* and the matrix $\boldsymbol{A}$ in (4.4) is called the **camera intrinsic matrix** [3].

In practical applications, the camera intrinsic parameters (or camera intrinsicmatrix) are normally constant and can be calibrated a priori—Presentation of knowledge on camera calibration will be postponed to Chap. 5—In contrast, the camera extrinsic parameters tend to vary during operation of the computer vision system and may be on-line estimated or adjusted a posteriori.

4.2.3 Perspective Mapping

Formalize (4.1) equivalently as

$$\begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{bmatrix} = [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} \tag{4.6}$$

or compactly as

$$\boldsymbol{P}_{\mathrm{c}} = [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w}}^{A},$$

where $\boldsymbol{P}_{\mathrm{w}}^{A} \equiv [x_{\mathrm{w}} \quad y_{\mathrm{w}} \quad z_{\mathrm{w}} \quad 1]^{\mathrm{T}}$ denotes the augmented vector of the 3D world point $\boldsymbol{P}_{\mathrm{w}}$.

By concatenating (4.6) and (4.4), one obtains the transform between the world coordinates system and the image coordinates system, formalized as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ [-1mm]y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ [-1mm]1 \end{bmatrix} \tag{4.7}$$

or compactly as

$$\lambda \boldsymbol{P}_{\mathrm{i}}^{A} = \boldsymbol{A}[\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w}}^{A},$$

which describes the **perspective mapping** from a generic 3D world point to its image projection. The concatenated matrix

$$\boldsymbol{M} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ [-1mm]0 & \beta & v_0 \\ [-1mm]0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \equiv \boldsymbol{A}[\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]$$

is called the *perspective mapping matrix*.

Instead of always treating the perspective mapping matrix as concatenation of the rotation-translation matrix and the camera intrinsic matrix, one may calibrate and use the perspective mapping matrix holistically in some applications [4–6].

4.2.4 Revisit Common Sense of Projective Geometry

After establishing algebraic foundations of projective geometry in the context of computer vision, the author revisits some aspects of common sense of projective geometry and see how they can be proved rigorously in algebraic way.

Basic invariance in projective geometry

Consider the four aspects of invariance presented in Sect. 4.1. For the *Invariance 1*, denote the perspective mapping matrix $\boldsymbol{M}$ in details as

$$\boldsymbol{M} \equiv \begin{bmatrix} \boldsymbol{M}_{1,:} \\ \boldsymbol{M}_{2,:} \\ \boldsymbol{M}_{3,:} \end{bmatrix} \equiv \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}.$$

Given a generic 3D world point $\boldsymbol{P}_{\mathrm{w}} \equiv [x_{\mathrm{w}} \quad y_{\mathrm{w}} \quad z_{\mathrm{w}}]^{\mathrm{T}}$, from (4.7) one can derive the image coordinates

$$\begin{aligned} u &= \frac{\boldsymbol{M}_{1,:}\boldsymbol{P}_{\mathrm{w}}^{A}}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^{A}} = \frac{m_{11}x_{\mathrm{w}}+m_{12}y_{\mathrm{w}}+m_{13}z_{\mathrm{w}}+m_{14}}{m_{31}x_{\mathrm{w}}+m_{32}y_{\mathrm{w}}+m_{33}z_{\mathrm{w}}+m_{34}}, \\ v &= \frac{\boldsymbol{M}_{2,:}\boldsymbol{P}_{\mathrm{w}}^{A}}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^{A}} = \frac{m_{21}x_{\mathrm{w}}+m_{22}y_{\mathrm{w}}+m_{23}z_{\mathrm{w}}+m_{24}}{m_{31}x_{\mathrm{w}}+m_{32}y_{\mathrm{w}}+m_{33}z_{\mathrm{w}}+m_{34}}, \end{aligned}$$

which are uniquely determined by the coordinates of the 3D world point. In other words, a 3D world point corresponds to only one image point, which implies that a point after perspective mapping is still a point. Thus the *Invariance 1* holds true.

For the *Invariance 2*, suppose a 3D world line L_{w} is parametrized in terms of t as

$$L_{\mathrm{w}}: \quad \boldsymbol{P}_{\mathrm{w}}(t) \equiv [x_{\mathrm{w}}^{0} + x_{\mathrm{w}}t \quad y_{\mathrm{w}}^{0} + y_{\mathrm{w}}t \quad z_{\mathrm{w}}^{0} + z_{\mathrm{w}}t]^{\mathrm{T}}.$$

Then

$$\begin{aligned} u(t) &= \frac{\boldsymbol{M}_{1,:}\boldsymbol{P}_{\mathrm{w}}^{A}(t)}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^{A}(t)} = \frac{m_{11}(x_{\mathrm{w}}^{0}+x_{\mathrm{w}}t)+m_{12}(y_{\mathrm{w}}^{0}+y_{\mathrm{w}}t)+m_{13}(z_{\mathrm{w}}^{0}+z_{\mathrm{w}}t)+m_{14}}{m_{31}(x_{\mathrm{w}}^{0}+x_{\mathrm{w}}t)+m_{32}(y_{\mathrm{w}}^{0}+y_{\mathrm{w}}t)+m_{33}(z_{\mathrm{w}}^{0}+z_{\mathrm{w}}t)+m_{34}} \\ &= \frac{(m_{11}x_{\mathrm{w}}^{0}+m_{12}y_{\mathrm{w}}^{0}+m_{13}z_{\mathrm{w}}^{0}+m_{14})+(m_{11}x_{\mathrm{w}}+m_{12}y_{\mathrm{w}}+m_{13}z_{\mathrm{w}})t}{(m_{31}x_{\mathrm{w}}^{0}+m_{32}y_{\mathrm{w}}^{0}+m_{33}z_{\mathrm{w}}^{0}+m_{34})+(m_{31}x_{\mathrm{w}}+m_{32}y_{\mathrm{w}}+m_{33}z_{\mathrm{w}})t} \\ &\equiv \frac{a_1+a_2t}{c_1+c_2t}, \\ v(t) &= \frac{\boldsymbol{M}_{2,:}\boldsymbol{P}_{\mathrm{w}}^{A}(t)}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^{A}(t)} = \frac{m_{21}(x_{\mathrm{w}}^{0}+x_{\mathrm{w}}t)+m_{22}(y_{\mathrm{w}}^{0}+y_{\mathrm{w}}t)+m_{23}(z_{\mathrm{w}}^{0}+z_{\mathrm{w}}t)+m_{24}}{m_{31}(x_{\mathrm{w}}^{0}+x_{\mathrm{w}}t)+m_{32}(y_{\mathrm{w}}^{0}+y_{\mathrm{w}}t)+m_{33}(z_{\mathrm{w}}^{0}+z_{\mathrm{w}}t)+m_{34}} \\ &= \frac{(m_{21}x_{\mathrm{w}}^{0}+m_{22}y_{\mathrm{w}}^{0}+m_{23}z_{\mathrm{w}}^{0}+m_{24})+(m_{21}x_{\mathrm{w}}+m_{22}y_{\mathrm{w}}+m_{23}z_{\mathrm{w}})t}{(m_{31}x_{\mathrm{w}}^{0}+m_{32}y_{\mathrm{w}}^{0}+m_{33}z_{\mathrm{w}}^{0}+m_{34})+(m_{31}x_{\mathrm{w}}+m_{32}y_{\mathrm{w}}+m_{33}z_{\mathrm{w}})t} \\ &\equiv \frac{b_1+b_2t}{c_1+c_2t}, \end{aligned}$$

where

$$\begin{aligned} a_1 &\equiv m_{11}x_{\mathrm{w}}^{0} + m_{12}y_{\mathrm{w}}^{0} + m_{13}z_{\mathrm{w}}^{0} + m_{14}, \quad & a_2 \equiv m_{11}x_{\mathrm{w}} + m_{12}y_{\mathrm{w}} + m_{13}z_{\mathrm{w}}, \\ b_1 &\equiv m_{21}x_{\mathrm{w}}^{0} + m_{22}y_{\mathrm{w}}^{0} + m_{23}z_{\mathrm{w}}^{0} + m_{24}, \quad & b_2 \equiv m_{21}x_{\mathrm{w}} + m_{22}y_{\mathrm{w}} + m_{23}z_{\mathrm{w}}, \\ c_1 &\equiv m_{31}x_{\mathrm{w}}^{0} + m_{32}y_{\mathrm{w}}^{0} + m_{33}z_{\mathrm{w}}^{0} + m_{34}, \quad & c_2 \equiv m_{31}x_{\mathrm{w}} + m_{32}y_{\mathrm{w}} + m_{33}z_{\mathrm{w}}. \end{aligned}$$

Take coefficients μ_1 and μ_2 satisfying

$$\begin{cases} a_1\mu_1 + b_1\mu_2 &= c_1 \\ a_2\mu_1 + b_2\mu_2 &= c_2 \end{cases}$$

to make up the *line equation* form

$$\mu_1 u(t) + \mu_2 v(t) = \frac{(a_1\mu_1+b_1\mu_2)+(a_2\mu_1+b_2\mu_2)t}{c_1+c_2t} = 1, \tag{4.8}$$

where μ_1 and μ_2 can be determined by the *Cramer rule* [7] as

$$\mu_1 = \frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, \quad \mu_2 = \frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}.$$

The line equation form (4.8) can exist if and only if the determinant

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1 b_2 - a_2 b_1 \neq 0.$$

But if $a_1 b_2 - a_2 b_1 = 0$, then

$$b_1 u(t) - a_1 v(t) = \frac{(b_1 a_1 - a_1 b_1) + (b_1 a_2 - a_1 b_2) t}{c_1 + c_2 t} = 0, \tag{4.9}$$

which is still a line equation form. Thus the *Invariance 2* holds true.

Based on the *Invariance 1* and *Invariance 2*, the *Invariance 3* is obvious, only if parallel lines are treated as coinciding at an imagined infinity point. The *Invariance 4* is also obvious, only if a point is treated as a degenerate line.

Vanishing point

Why do a group of parallel lines lead to a vanishing point? Consider a group of parallel 3D world lines parametrized generically as

$$L_\mathrm{w}: \ \boldsymbol{P}_\mathrm{w}(t) \equiv \begin{bmatrix} x_\mathrm{w}^0 + x_\mathrm{w} t & y_\mathrm{w}^0 + y_\mathrm{w} t & z_\mathrm{w}^0 + z_\mathrm{w} t \end{bmatrix}^\mathrm{T},$$

where the direction vector

$$\boldsymbol{d} \equiv \begin{bmatrix} x_\mathrm{w} & y_\mathrm{w} & z_\mathrm{w} \end{bmatrix}^\mathrm{T}$$

is fixed but the starting point

$$\boldsymbol{P}_\mathrm{w}^0 \equiv \begin{bmatrix} x_\mathrm{w}^0 & y_\mathrm{w}^0 & z_\mathrm{w}^0 \end{bmatrix}^\mathrm{T} \in G$$

is changeable and depends on each concrete line in the group.

Then one has

$$\begin{aligned} u(t) &= \frac{\boldsymbol{M}_{1,:} \boldsymbol{P}_\mathrm{w}^A(t)}{\boldsymbol{M}_{3,:} \boldsymbol{P}_\mathrm{w}^A(t)} \\ &= \frac{(m_{11} x_\mathrm{w}^0 + m_{12} y_\mathrm{w}^0 + m_{13} z_\mathrm{w}^0 + m_{14}) + (m_{11} x_\mathrm{w} + m_{12} y_\mathrm{w} + m_{13} z_\mathrm{w}) t}{(m_{31} x_\mathrm{w}^0 + m_{32} y_\mathrm{w}^0 + m_{33} z_\mathrm{w}^0 + m_{34}) + (m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}) t}, \\ v(t) &= \frac{\boldsymbol{M}_{2,:} \boldsymbol{P}_\mathrm{w}^A(t)}{\boldsymbol{M}_{3,:} \boldsymbol{P}_\mathrm{w}^A(t)} \\ &= \frac{(m_{21} x_\mathrm{w}^0 + m_{22} y_\mathrm{w}^0 + m_{23} z_\mathrm{w}^0 + m_{24}) + (m_{21} x_\mathrm{w} + m_{22} y_\mathrm{w} + m_{23} z_\mathrm{w}) t}{(m_{31} x_\mathrm{w}^0 + m_{32} y_\mathrm{w}^0 + m_{33} z_\mathrm{w}^0 + m_{34}) + (m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}) t}, \end{aligned}$$

As $t \to \infty$, the $\begin{bmatrix} u(t) & v(t) \end{bmatrix}^\mathrm{T}$ of all the parallel lines approach the common limit

$$\begin{aligned} u(t) &\to \frac{m_{11} x_\mathrm{w} + m_{12} y_\mathrm{w} + m_{13} z_\mathrm{w}}{m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}} \\ v(t) &\to \frac{m_{21} x_\mathrm{w} + m_{22} y_\mathrm{w} + m_{23} z_\mathrm{w}}{m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}} \end{aligned}$$

regardless of their concrete starting points $\boldsymbol{P}_\mathrm{w}^0$ respectively. The common limit

$$\begin{bmatrix} \frac{m_{11} x_\mathrm{w} + m_{12} y_\mathrm{w} + m_{13} z_\mathrm{w}}{m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}} & \frac{m_{21} x_\mathrm{w} + m_{22} y_\mathrm{w} + m_{23} z_\mathrm{w}}{m_{31} x_\mathrm{w} + m_{32} y_\mathrm{w} + m_{33} z_\mathrm{w}} \end{bmatrix}^\mathrm{T}$$

is right their vanishing point.

Cross-ratio

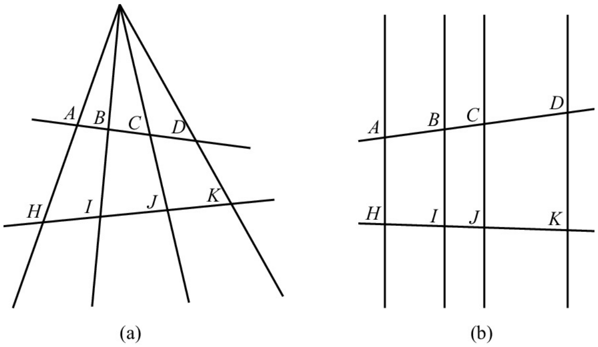
As illustrated in Fig. 4.8a, given four lines that coincide at the same point and given two arbitrary other lines that intersect the four coinciding lines at $\{A, B, C, D\}$ and $\{H, I, J, K\}$ respectively, the equality

$$\frac{|AC|\cdot|BD|}{|AD|\cdot|BC|} = \frac{|HJ|\cdot|IK|}{|HK|\cdot|IJ|} \tag{4.10}$$

always holds. In other words, given an arbitrary line that intersects the four coinciding lines at $\{A, B, C, D\}$, then the ratio

$$\frac{|AC|\cdot|BD|}{|AD|\cdot|BC|} \tag{4.11}$$

is invariant (constant), regardless of the choice of the intersecting line *A*–*B*–*C*–*D*. The ratio defined in (4.11) is called the *cross-ratio* of the four coinciding lines.



(a) (b)

Fig. 4.8 Cross-ratio **a** original configuration imagined as projected image; **b** imagined 3D world scenario of parallel lines

In the spirit of projective geometry, imagine the four coinciding lines as projected image of four parallel lines in the 3D world, and let the four parallel lines be located on the $x_\mathrm{w} - y_\mathrm{w}$ plane in the world coordinates system, as illustrated in the right sub-figure of Fig. 4.8. Suppose the points *A*, *B*, *C*, and *D* in the 3D world are denoted as

$$\begin{array}{rl} & A: (x_0 + x_\mathrm{w} t_a, y_0 + y_\mathrm{w} t_a, 0) \\ & B: (x_0 + x_\mathrm{w} t_b, y_0 + y_\mathrm{w} t_b, 0) \\ *[-2.5pt] & C: (x_0 + x_\mathrm{w} t_c, y_0 + y_\mathrm{w} t_c, 0) \\ *[-2.5pt] & D: (x_0 + x_\mathrm{w} t_d, y_0 + y_\mathrm{w} t_d, 0) \end{array}$$

where the ratios among $t_b - t_a$, $t_c - t_a$, $t_d - t_a$, $t_c - t_b$, $t_d - t_b$, and $t_d - t_c$ are apparently constant.

From (4.7) one can derive the image coordinates of *A*, *B*, *C*, and *D*:

$$\begin{aligned} u_i &= \tfrac{m_{11}(x_0+x_\mathrm{w}t_i)+m_{12}(y_0+y_\mathrm{w}t_i)+m_{14}}{m_{31}(x_0+x_\mathrm{w}t_i)+m_{32}(y_0+y_\mathrm{w}t_i)+m_{34}} = \tfrac{a_1+a_2t_i}{c_1+c_2t_i}, \\ v_i &= \tfrac{m_{21}(x_0+x_\mathrm{w}t_i)+m_{22}(y_0+y_\mathrm{w}t_i)+m_{24}}{m_{31}(x_0+x_\mathrm{w}t_i)+m_{32}(y_0+y_\mathrm{w}t_i)+m_{34}} = \tfrac{b_1+b_2t_i}{c_1+c_2t_i}, \end{aligned}$$

where $i \in \{a, b, c, d\}$ and

$$\begin{aligned} a_1 &\equiv m_{11}x_0 + m_{12}y_0 + m_{14}, \quad a_2 \equiv m_{11}x_{\mathrm{w}} + m_{12}y_{\mathrm{w}}, \\ b_1 &\equiv m_{21}x_0 + m_{22}y_0 + m_{24}, \quad b_2 \equiv m_{21}x_{\mathrm{w}} + m_{22}y_{\mathrm{w}}, \\ c_1 &\equiv m_{31}x_0 + m_{32}y_0 + m_{34}, \quad c_2 \equiv m_{31}x_{\mathrm{w}} + m_{32}y_{\mathrm{w}}. \end{aligned}$$

For $\forall i, j \in \{a, b, c, d\}$, one has

$$\begin{aligned} u_i - u_j &= \tfrac{a_1+a_2t_i}{c_1+c_2t_i} - \tfrac{a_1+a_2t_j}{c_1+c_2t_j} = \tfrac{(a_2c_1-a_1c_2)(t_i-t_j)}{(c_1+c_2t_i)(c_1+c_2t_j)}, \\ v_i - v_j &= \tfrac{b_1+b_2t_i}{c_1+c_2t_i} - \tfrac{b_1+b_2t_j}{c_1+c_2t_j} = \tfrac{(b_2c_1-b_1c_2)(t_i-t_j)}{(c_1+c_2t_i)(c_1+c_2t_j)}. \end{aligned}$$

So

$$\tfrac{(u_c-u_a)(u_d-u_b)}{(u_d-u_a)(u_c-u_b)} = \frac{\dfrac{(a_2c_1 - a_1c_2)^2(t_c - t_a)(t_d - t_b)}{\Pi_{i\in\{a,b,c,d\}}(c_1 + c_2t_i)}}{\dfrac{(a_2c_1 - a_1c_2)^2(t_d - t_a)(t_c - t_b)}{\Pi_{i\in\{a,b,c,d\}}(c_1 + c_2t_i)}} = \tfrac{(t_c-t_a)(t_d-t_b)}{(t_d-t_a)(t_c-t_b)},$$

$$\tfrac{(v_c-v_a)(v_d-v_b)}{(v_d-v_a)(v_c-v_b)} = \frac{\dfrac{(b_2c_1 - b_1c_2)^2(t_c - t_a)(t_d - t_b)}{\Pi_{i\in\{a,b,c,d\}}(c_1 + c_2t_i)}}{\dfrac{(b_2c_1 - b_1c_2)^2(t_d - t_a)(t_c - t_b)}{\Pi_{i\in\{a,b,c,d\}}(c_1 + c_2t_i)}} = \tfrac{(t_c-t_a)(t_d-t_b)}{(t_d-t_a)(t_c-t_b)},$$

and the cross-ratio

$$\tfrac{|AC|\cdot|BD|}{|AD|\cdot|BC|} = \tfrac{(u_c-u_a)(u_d-u_b)}{(u_d-u_a)(u_c-u_b)} = \tfrac{(v_c-v_a)(v_d-v_b)}{(v_d-v_a)(v_c-v_b)} = \tfrac{(t_c-t_a)(t_d-t_b)}{(t_d-t_a)(t_c-t_b)} \tag{4.12}$$

is indeed invariant, regardless of the choice of (x_0, y_0) and $(x_{\mathrm{w}}, y_{\mathrm{w}})$, namely regardless of the choice of the intersecting line *A–B–C–D*.

Circle projected to ellipse
It is known that the projected image of an entire circle in the 3D world is an ellipse (including circle). To verify this, suppose the circle in the 3D world is parametrized in terms of θ as

$$C_{\mathrm{w}}: \quad \boldsymbol{P}_{\mathrm{w}}(\theta) \equiv [r\cos\theta \quad r\sin\theta \quad 0]^{\mathrm{T}},$$

for which the circle center is set to the origin of the world coordinates system for analysis convenience yet without loss of generality.

Via perspective mapping one has

$$\begin{aligned} u(\theta) &= \tfrac{\boldsymbol{M}_{1,:}\boldsymbol{P}_{\mathrm{w}}^A(\theta)}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^A(\theta)} = \tfrac{m_{11}r\cos\theta+m_{12}r\sin\theta+m_{14}}{m_{31}r\cos\theta+m_{32}r\sin\theta+m_{34}}, \\ v(\theta) &= \tfrac{\boldsymbol{M}_{2,:}\boldsymbol{P}_{\mathrm{w}}^A(\theta)}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\mathrm{w}}^A(\theta)} = \tfrac{m_{21}r\cos\theta+m_{22}r\sin\theta+m_{24}}{m_{31}r\cos\theta+m_{32}r\sin\theta+m_{34}}, \end{aligned}$$

which can be re-formalized as

$$\lambda \begin{bmatrix} u(\theta) \\ v(\theta) \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}r & m_{12}r & m_{14} \\ m_{21}r & m_{22}r & m_{24} \\ m_{31}r & m_{32}r & m_{34} \end{bmatrix} \begin{bmatrix} \cos\theta \\ \sin\theta \\ 1 \end{bmatrix}. \tag{4.13}$$

Neglect the singularity case where the projected image of the circle becomes degenerate to a line and suppose the inverse of the 3-by-3 matrix in (4.13) exists. Denote such inverse as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \equiv \begin{bmatrix} m_{11}r & m_{12}r & m_{14} \\ m_{21}r & m_{22}r & m_{24} \\ m_{31}r & m_{32}r & m_{34} \end{bmatrix}^{-1}.$$

From (4.13) one has

$$\begin{bmatrix} \cos\theta \\ \sin\theta \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

namely

$$\cos\theta = \tfrac{a_{11}u+a_{12}v+a_{13}}{a_{31}u+a_{32}v+a_{33}}, \tag{4.14}$$

$$\sin\theta = \tfrac{a_{21}u+a_{22}v+a_{23}}{a_{31}u+a_{32}v+a_{33}}, \tag{4.15}$$

where explicit expression of the parameter θ is omitted. So

$$\begin{aligned} & \left(\tfrac{a_{11}u+a_{12}v+a_{13}}{a_{31}u+a_{32}v+a_{33}}\right)^2 + \left(\tfrac{a_{21}u+a_{22}v+a_{23}}{a_{31}u+a_{32}v+a_{33}}\right)^2 = 1 \\ \Leftrightarrow\ & (a_{11}u + a_{12}v + a_{13})^2 + (a_{21}u + a_{22}v + a_{23})^2 = (a_{31}u + a_{32}v + a_{33})^2 \\ \Leftrightarrow\ & b_{11}u^2 + b_{22}v^2 + 2b_{12}uv + 2b_1u + 2b_2v + b_0 = 0, \end{aligned} \tag{4.16}$$

where

$$\begin{aligned} & b_{11} \equiv a_{11}^2 + a_{21}^2 - a_{31}^2, b_{22} \equiv a_{12}^2 + a_{22}^2 - a_{32}^2, b_{12} \equiv a_{11}a_{12} + a_{21}a_{22} - a_{31}a_{32}, \\ & b_1 \equiv a_{11}a_{13} + a_{21}a_{23} - a_{31}a_{33}, b_2 \equiv a_{12}a_{13} + a_{22}a_{23} - a_{32}a_{33}, b_0 \equiv a_{13}^2 + a_{23}^2 - a_{33}^2. \end{aligned}$$

On one hand, any point (u, v) of the projected image must be located on the quadratic curve described by (4.16). On the other hand, as (4.13) and (4.14) imply a bijective relationship exists between θ and (u, v), any (u, v) satisfying (4.16) must have a corresponding θ. In other words, any point of the quadratic curve described by (4.16) must be the projection of a point of the circle. Therefore, the projected image of the circle is right the quadratic curve described by (4.16), which is an ellipse, a parabola, or a hyperbola.

Since the circle is closed and bounded and perspective mapping is continuous, the projected image of the entire circle must also be bounded, which implies that the projected image of the circle cannot be a parabola or hyperbola, but can only be an ellipse

—It is worth noting that if one did not suppose to be able to have the projection image of the entire circle, (i.e. if one did not exclude the degenerate case that some part of the circle has no projection image), then the judgement that the projection image is bounded would not hold.

4.3 Homography

4.3.1 Bijective Mapping Between Two-Dimensional Planes

The perspective mapping model is formalized in (4.7):

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{\text{wc}} & \boldsymbol{T}_{\text{wc}} \end{bmatrix} \begin{bmatrix} x_{\text{w}} \\ y_{\text{w}} \\ z_{\text{w}} \\ 1 \end{bmatrix},$$

from which one can uniquely determine the image projection of a generic 3D world point. More specifically, denote the perspective mapping matrix in details as

$$\boldsymbol{M} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ {[-1mm]}0 & \beta & v_0 \\ {[-1mm]}0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{\text{wc}} & \boldsymbol{T}_{\text{wc}} \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{M}_{1,:} \\ {[-1mm]}\boldsymbol{M}_{2,:} \\ {[-1mm]}\boldsymbol{M}_{3,:} \end{bmatrix} \equiv \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ {[-1mm]}m_{21} & m_{22} & m_{23} & m_{24} \\ {[-1mm]}m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}.$$

Then given a generic 3D world point $\boldsymbol{P}_{\text{w}} \equiv \begin{bmatrix} x_{\text{w}} & y_{\text{w}} & z_{\text{w}} \end{bmatrix}^{\text{T}}$, from (4.7) one can derive the image coordinates

$$u = \frac{\boldsymbol{M}_{1,:}\boldsymbol{P}_{\text{w}}^{A}}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\text{w}}^{A}} = \frac{m_{11}x_{\text{w}}+m_{12}y_{\text{w}}+m_{13}z_{\text{w}}+m_{14}}{m_{31}x_{\text{w}}+m_{32}y_{\text{w}}+m_{33}z_{\text{w}}+m_{34}}, \tag{4.17a}$$

$$v = \frac{\boldsymbol{M}_{2,:}\boldsymbol{P}_{\text{w}}^{A}}{\boldsymbol{M}_{3,:}\boldsymbol{P}_{\text{w}}^{A}} = \frac{m_{21}x_{\text{w}}+m_{22}y_{\text{w}}+m_{23}z_{\text{w}}+m_{24}}{m_{31}x_{\text{w}}+m_{32}y_{\text{w}}+m_{33}z_{\text{w}}+m_{34}}, \tag{4.17b}$$

which naturally implies uniqueness of the image projection.

:

$$(\boldsymbol{M}_{1,:} - u\boldsymbol{M}_{3,:})\boldsymbol{P}_{\text{w}}^{A} = 0, \tag{4.18a}$$

$$(\boldsymbol{M}_{2,:} - v\boldsymbol{M}_{3,:})\boldsymbol{P}_{\text{w}}^{A} = 0, \tag{4.18b}$$

namely in details as

$$\begin{aligned} (m_{11}-u\cdot m_{31})x_{\text{w}}+(m_{12}-u\cdot m_{32})y_{\text{w}}+(m_{13}-u\cdot m_{33})z_{\text{w}}+m_{14}-u\cdot m_{34} &= 0, \\ (m_{21}-v\cdot m_{31})x_{\text{w}}+(m_{22}-v\cdot m_{32})y_{\text{w}}+(m_{23}-v\cdot m_{33})z_{\text{w}}+m_{24}-v\cdot m_{34} &= 0. \end{aligned}$$

There are three unknown
variables $(x_{\text{w}}, y_{\text{w}}, z_{\text{w}})$ in contrast with only two equations in (4.18), so (4.18) does not have a unique solution of $(x_{\text{w}}, y_{\text{w}}, z_{\text{w}})$, which implies that the image point $\boldsymbol{P}_{\text{i}}$ does not correspond to a unique 3D world counterpart but an infinite number of 3D world counterparts.[4]

As demonstrated above, the relationship between the world coordinates system and the image coordinates system is not bijective.

Now constrain consideration of 3D world points to a target plane in the 3D world. Without loss of generality, suppose the world coordinates system is established with its $O_{\mathrm{w}} - X_{\mathrm{w}} - Y_{\mathrm{w}}$ plane located on the target plane, as illustrated in Fig. 4.9. Given a generic 3D world point $\boldsymbol{P}_{\mathrm{wp}} \equiv [x_{\mathrm{w}} \quad y_{\mathrm{w}}]^{\mathrm{T}}$ on the target plane (the third world coordinate $z_{\mathrm{w}} \equiv 0$ is omitted), then (4.7) is reduced to[5]

$$\begin{aligned} \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \mathbf{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}} \mathbf{e}_2 \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix} \end{aligned} \tag{4.19}$$

or compactly as

$$\lambda \boldsymbol{P}_{\mathrm{i}}^{\mathrm{A}} = \boldsymbol{H} \boldsymbol{P}_{\mathrm{wp}}^{\mathrm{A}},$$

where

$$\boldsymbol{H} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \boldsymbol{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}} \boldsymbol{e}_2 \quad \boldsymbol{T}_{\mathrm{wc}}].$$

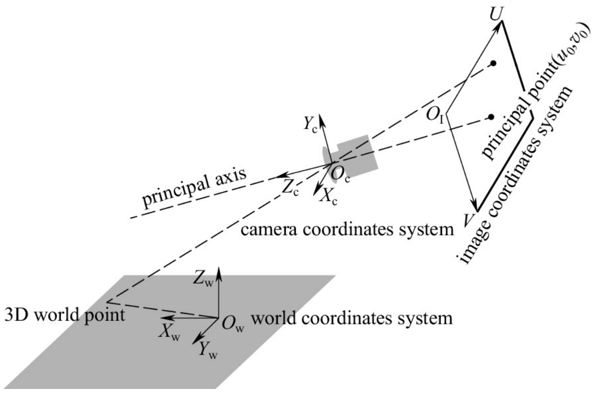

U
principal point(u_0,v_0)
O_{i}
image coordinates system
V
Y_{c}
O_{c}
principal axis
Z_{c}
X_{c}
camera coordinates system
Z_{w}
3D world point
O_{w}
world coordinates system
X_{w}
Y_{w}

Fig. 4.9 Homography

In practical applications, the camera intrinsic matrix

$$\begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

is invertible. Besides, the target plane spanned by the vectors $\boldsymbol{R}_{\mathrm{wc}} \boldsymbol{e}_1$ and $\boldsymbol{R}_{\mathrm{wc}} \boldsymbol{e}_2$ is normally not aligned with the vector $\boldsymbol{T}_{\mathrm{wc}}$, and hence the matrix

$$[\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2 \quad \boldsymbol{T}_{\mathrm{wc}}]$$

is also invertible. So the matrix $\boldsymbol{H}$ in is normally invertible. Then

$$\frac{1}{\lambda}\begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix} = [\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2 \quad \boldsymbol{T}_{\mathrm{wc}}]^{-1} \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.20}$$

or compactly as

$$\frac{1}{\lambda}\boldsymbol{P}_{\mathrm{wp}}^{A} = \boldsymbol{H}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A}.$$

The formalism duality between (4.19) and (4.20) reflects a bijective relationship between the target plane and the image plane. In other words, given a generic point $\boldsymbol{P}_{\mathrm{wp}} \equiv [x_{\mathrm{w}} \quad y_{\mathrm{w}}]^{\mathrm{T}}$ on the target plane, the corresponding image point $\boldsymbol{P}_{\mathrm{i}}$ can be uniquely determined as

$$u = \frac{\boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{H}\boldsymbol{P}_{\mathrm{wp}}^{A}}{\boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{H}\boldsymbol{P}_{\mathrm{wp}}^{A}}, \tag{4.21a}$$

$$v = \frac{\boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{H}\boldsymbol{P}_{\mathrm{wp}}^{A}}{\boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{H}\boldsymbol{P}_{\mathrm{wp}}^{A}}. \tag{4.21b}$$

On the other hand, given a generic image point $\boldsymbol{P}_{\mathrm{i}}$, the corresponding 3D world point $\boldsymbol{P}_{\mathrm{wp}}$ on the target plane can be uniquely determined as

$$x_w = \frac{\boldsymbol{e}_1^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{P}_{\mathrm{i}}^{\mathrm{A}}}{\boldsymbol{e}_3^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{P}_{\mathrm{i}}^{\mathrm{A}}}, \tag{4.22a}$$

$$y_w = \frac{\boldsymbol{e}_2^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{P}_{\mathrm{i}}^{\mathrm{A}}}{\boldsymbol{e}_3^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{P}_{\mathrm{i}}^{\mathrm{A}}}. \tag{4.22b}$$

The bijective relationship between the target plane and the image plane is coined as **homography**. The invertible matrix $\boldsymbol{H}$ in (4.19), which characterizes the bijective relationship between the two planes, is called the *homography matrix*.

Matlab code for performing homography, namely computing the image projections of the points on the target plane via (4.21), is given as follows.

HomographyXY2UV.m

```
%% Homography from the target plane to the image plane
% Hm: homography matrix
% xw: x-coordinates of a group of points on the 3D world target
    plane
% yw: y-coordinates of a group of points on the 3D world target
    plane
% u : u-coordinates of their image projections
% v : v-coordinates of their image projections
function [u, v] = HomographyXY2UV(Hm, xw, yw)
    xyAug = [reshape(xw,1,[]); reshape(yw,1,[]); ones(1,length(
    xw))];
    uvAug = Hm*xyAug;
    u = uvAug(1,:)./uvAug(3,:); v = uvAug(2,:)./uvAug(3,:);
end
```

4.3.2 Homography Calibration

If the camera intrinsic parameters $\{\alpha, \beta, \gamma, u_0, v_0\}$ and the camera extrinsic parameters $\{\boldsymbol{R}_{\text{wc}}, \boldsymbol{T}_{\text{wc}}\}$ are known, then the homography matrix $\boldsymbol{H}$ that characterizes the bijective relationship between the target plane and the image plane can be directly derived.

However, camera extrinsic parameters are usually unknown in practical applications; sometimes, even camera intrinsic parameters are unknown either. A question arises naturally: How to know the homography matrix in absence of camera intrinsic and extrinsic parameters ?

For this purpose, one may resort to *control points* namely a set of 3D world points whose world coordinates and corresponding image coordinates are registered or can be known. Then one can reveal the homography matrix by analysing such world and image coordinates. This way of knowing the homography matrix is called *homography calibration*.

More specifically, given a set of 3D world points $\boldsymbol{P}_{\text{wp},1}$, $\boldsymbol{P}_{\text{wp},2}$, ..., $\boldsymbol{P}_{\text{wp},n}$ whose world coordinates

$$\boldsymbol{P}_{\text{wp},1} \equiv \begin{bmatrix} x_{\text{w},1} \\ y_{\text{w},1} \end{bmatrix}, \quad \boldsymbol{P}_{\text{wp},2} \equiv \begin{bmatrix} x_{\text{w},2} \\ y_{\text{w},2} \end{bmatrix}, \quad \cdots \quad , \quad \boldsymbol{P}_{\text{wp},n} \equiv \begin{bmatrix} x_{\text{w},n} \\ y_{\text{w},n} \end{bmatrix}$$

are known and the image coordinates of their image projections

$$\boldsymbol{P}_{\text{i},1} \equiv \begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \quad \boldsymbol{P}_{\text{i},2} \equiv \begin{bmatrix} u_2 \\ v_2 \end{bmatrix}, \quad \cdots \quad , \quad \boldsymbol{P}_{\text{i},n} \equiv \begin{bmatrix} u_n \\ v_n \end{bmatrix}$$

are known as well.

For each pair of $\boldsymbol{P}_{\text{wp},k}$ and $\boldsymbol{P}_{\text{i},k}$ $(k = 1, \ldots, n)$, (4.21) gives

$$\begin{aligned} u_k &= \frac{\boldsymbol{e}_1^{\text{T}} \boldsymbol{H} \boldsymbol{P}_{\text{wp},k}^{A}}{\boldsymbol{e}_3^{\text{T}} \boldsymbol{H} \boldsymbol{P}_{\text{wp},k}^{A}}, \\ v_k &= \frac{\boldsymbol{e}_2^{\text{T}} \boldsymbol{H} \boldsymbol{P}_{\text{wp},k}^{A}}{\boldsymbol{e}_3^{\text{T}} \boldsymbol{H} \boldsymbol{P}_{\text{wp},k}^{A}}, \end{aligned}$$

and further gives two equations in terms of the homography matrix $\boldsymbol{H}$:

$$\begin{bmatrix} (\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} & \mathbf{0} & -u_k(\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} \\ \mathbf{0} & (\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} & -v_k(\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_1 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_2 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_3 \end{bmatrix} = \mathbf{0}.$$

So there are a total number of $2n$ equations

$$\begin{bmatrix} (\boldsymbol{P}^A_{\mathrm{wp},1})^{\mathrm{T}} & \mathbf{0} & -u_1(\boldsymbol{P}^A_{\mathrm{wp},1})^{\mathrm{T}} \\ \mathbf{0} & (\boldsymbol{P}^A_{\mathrm{wp},1})^{\mathrm{T}} & -v_1(\boldsymbol{P}^A_{\mathrm{wp},1})^{\mathrm{T}} \\ \vdots & \vdots & \vdots \\ (\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} & \mathbf{0} & -u_k(\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} \\ \mathbf{0} & (\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} & -v_k(\boldsymbol{P}^A_{\mathrm{wp},k})^{\mathrm{T}} \\ \vdots & \vdots & \vdots \\ (\boldsymbol{P}^A_{\mathrm{wp},n})^{\mathrm{T}} & \mathbf{0} & -u_n(\boldsymbol{P}^A_{\mathrm{wp},n})^{\mathrm{T}} \\ \mathbf{0} & (\boldsymbol{P}^A_{\mathrm{wp},n})^{\mathrm{T}} & -v_n(\boldsymbol{P}^A_{\mathrm{wp},n})^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_1 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_2 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_3 \end{bmatrix} = \mathbf{0}. \tag{4.23}$$

Denote the homography matrix $\boldsymbol{H}$ in details as

$$\boldsymbol{H} \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

Then (4.23) is expressed in details as

$$\begin{bmatrix} x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & 0 & 0 & 0 & -u_1x_{\mathrm{w},1} & -u_1y_{\mathrm{w},1} & -u_1 \\ 0 & 0 & 0 & x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & -v_1x_{\mathrm{w},1} & -v_1y_{\mathrm{w},1} & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{\mathrm{w},k} & y_{\mathrm{w},k} & 1 & 0 & 0 & 0 & -u_kx_{\mathrm{w},k} & -u_ky_{\mathrm{w},k} & -u_k \\ 0 & 0 & 0 & x_{\mathrm{w},k} & y_{\mathrm{w},k} & 1 & -v_kx_{\mathrm{w},k} & -v_ky_{\mathrm{w},k} & -v_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{\mathrm{w},n} & y_{\mathrm{w},n} & 1 & 0 & 0 & 0 & -u_nx_{\mathrm{w},n} & -u_ny_{\mathrm{w},n} & -u_n \\ 0 & 0 & 0 & x_{\mathrm{w},n} & y_{\mathrm{w},n} & 1 & -v_nx_{\mathrm{w},n} & -v_ny_{\mathrm{w},n} & -v_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \mathbf{0}. \tag{4.24}$$

Denote the 9-dimensional column vector in (4.23) and (4.24) compactly as

$$\boldsymbol{h}_{\mathrm{r}} \equiv \begin{bmatrix} \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_1 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_2 \\ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{e}_3 \end{bmatrix} \equiv \begin{bmatrix} h_{11} \\ \vdots \\ h_{33} \end{bmatrix}$$

and denote the 2*n*-by-9 matrix in (4.23) and (4.24) compactly as $\boldsymbol{L}_{2n,9}$. Then the *homogeneous linear equation* described by (4.23) and (4.24) is formalized compactly as

$$\boldsymbol{L}_{2n,9}\boldsymbol{h}_{\mathrm{r}} = \mathbf{0}. \tag{4.25}$$

The solution of $\boldsymbol{h}_{\mathrm{r}}$ of the homogeneous linear equation (4.25) is the normalized eigenvector associated with the smallest eigenvalue of $\boldsymbol{L}_{2n,9}^{\mathrm{T}}\boldsymbol{L}_{2n,9}$—refer to (F.14) in Sect. F.2 in Appendix F.

Once $\boldsymbol{h}_{\mathrm{r}}$ is solved, one can recover the homography matrix $\boldsymbol{H}$ by arranging elements of $\boldsymbol{h}_{\mathrm{r}}$ in corresponding positions of the homography matrix.

It is worth noting that the homography matrix has no influence at all on the plane-to-plane bijective relationship characterized by it if it changes only by a scaling factor. So it does not matter whether the normalized version of the relevant eigenvector is taken or not, yet the normalized version tends to be taken by convention.

Matlab code for homography calibration is given as follows.

CalibrateHomography.m

```
%% Calibrate the homography matrix according to control points
% Hm: homography matrix
% xwCP: x-coordinates of control points (CP) on the 3D world
    target plane
% ywCP: y-coordinates of control points (CP) on the 3D world
    target plane
% uCP : u-coordinates of their image projections
% vCP : v-coordinates of their image projections
function Hm = CalibrateHomography(uCP, vCP, xwCP, ywCP)
    nCP = length(uCP); L = zeros(2*nCP,9);
    for k = 1:nCP
        L(2*k,:) = [xwCP(k),ywCP(k),1,0,0,0,-uCP(k)*[xwCP(k),
    ywCP(k),1]];
        L(2*k+1,:) = [0,0,0,xwCP(k),ywCP(k),1,-vCP(k)*[xwCP(k),
    ywCP(k),1]];
    end
    hr = SolveHomogeneousLinearEquation(L);
    Hm = [hr(1), hr(2), hr(3); ...
             hr(4), hr(5), hr(6); ...
             hr(7), hr(8), hr(9)];
end
```

The involved sub-function **SolveHomogeneousLinearEquation.m** is given in Sect. F.2 in Appendix F.

4.3.3 Inverse Perspective Mapping

The perspective mapping determines how a generic 3D world point is projected to the image plane, but cannot determine how an image point is inversely projected to its 3D world counterpart. In contrast, the homography not only determines how a generic 3D world point on the target plane is projected to the image plane, but also determines how an image point is inversely projected to its 3D world counterpart on the target plane, in the light of (4.20) and (4.22).

Such inverse projection from the image plane to the target plane can be coined as **inverse perspective mapping**, which is often adopted to generate *normal views* or *bird-eye views* of relevant scenarios in practical applications [8–13].

For example, the inverse perspective mapping of a road scenario in the context of lane detection is illustrated in Fig. 4.10. The bird-eye view image of the lanes in the road scenario is generated. For lane detection, the main advantage of the bird-eye view image over the original camera image is that lane mark features in the bird-eye view image well reflect their actual geometric properties in the real world, so general a priori knowledge of lane geometry such as lane mark parallelism, lane width, lane mark length can be exploited to enhance robustness of lane detection.

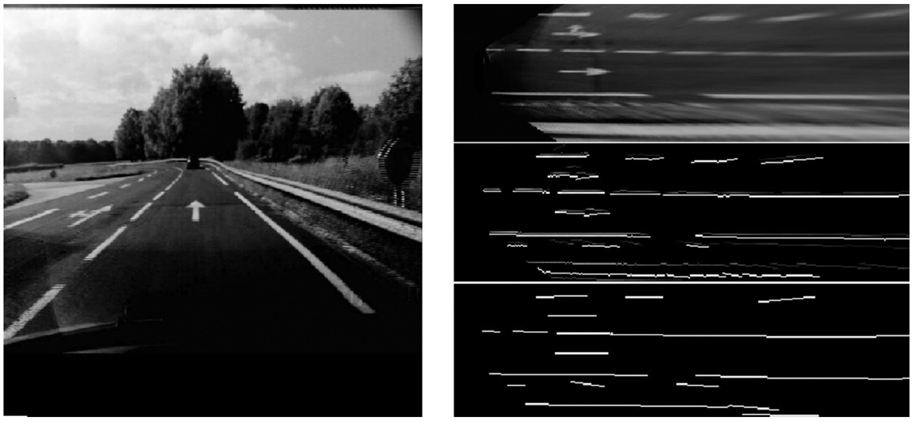
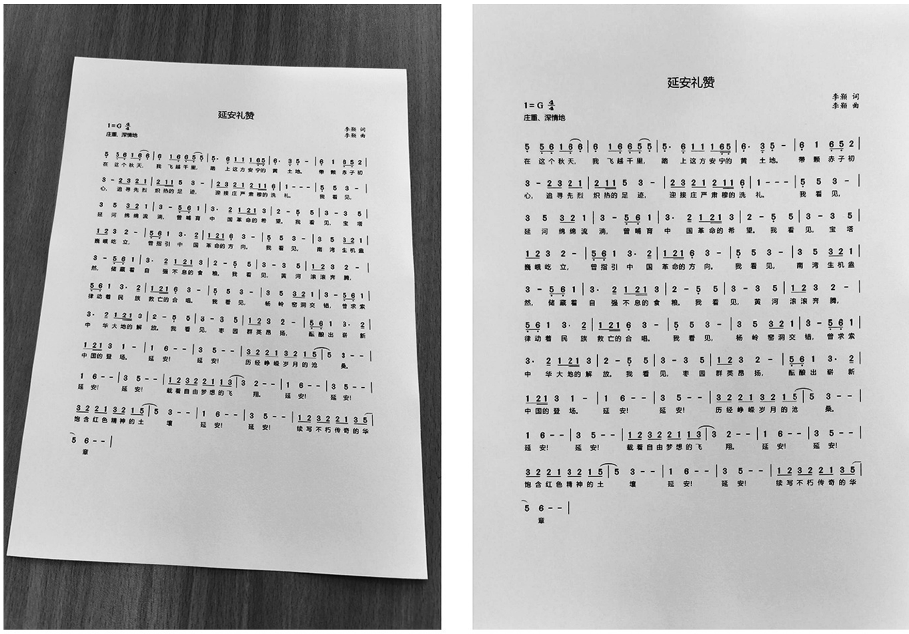


Fig. 4.10 Inverse perspective mapping for lane detection [4]

Inverse perspective mapping is not only useful to engineering activities, as demonstrated above. Flexible application of inverse perspective mapping can also facilitate daily life. For example, suppose one would like to scan a paper but has only a smart mobile phone at hand.[6] Although not having any scanner at hand to generate a scanned image of the paper, one can use the smart mobile phone to take a photo of the paper and then perform inverse perspective mapping on the photo to generate its normal view image version, as if she or he obtained a scanned image. Such effect of virtual file scanning is illustrated in Fig. 4.11.



(a) (b)

Fig. 4.11 Virtual file scanning **a** original camera image; **b** normal view image as if scanned

Figure 4.11a illustrates the original photo of the printed music score of “Tribute to Yan-An”.[7] The music score is printed on a A4 paper, the size of which is known a priori:

the A4 paper has a height of 29.7 cm and a width of 21.0 cm. The photo is captured by a *HUAWEI* smart mobile phone.

Figure 4.11b illustrates the normal view image of the A4 paper, which is obtained via inverse perspective mapping. For this purpose, manually select the four corners of the A4 paper on the image. In Matlab, the built-in function **ginput** enables such manual selection. In implementation by other programming languages, similar functions that provide cursor click locations also exist—In fact, the Harris corner detection method presented in Sect. 2.4 in Chap. 2 can be adopted to facilitate extraction of the image coordinates of the four corners. So in practice, one does not need to scrutinize the cursor on the screen for sake of clicking exactly on the four corners; one can just click nearby the four corners and then detected Harris corners are matched automatically—The world coordinates of the four corners on the paper plane are already known a priori, namely (in the metric unit of meter)

$$(0.000, 0.000), \quad (0.210, 0.000), \quad (0.210, 0.297), \quad (0.000, 0.297).$$

Substitute the a priori world coordinates and the extracted image coordinates of the four corners into (4.24) to calibrate the homography between the paper plane and the image plane.

Once the homography matrix is computed, the original camera image can be inversely projected onto the paper plane to form the normal view image of the A4 paper, as illustrated in Fig. 4.11b. The normal view image obtained via inverse perspective mapping of the original camera image forms a vivid “virtual scanning” effect as if it is generated directly by a real scanner.[8]

Matlab code for demonstration of virtual file scanning is given as follows.

demoVirtualFileScanning.m

```
%% Virtual File Scanning
imgo = imread(’.\tribute_to_yanan.jpg’);
img = rgb2gray(imgo); imgDb = double(img);
[ptHC, iHC] = CornerHarris(img, 66666); uHC = ptHC(1,:); vHC =
    ptHC(2,:);
figure(1), imshow(imgo); hold on; plot(uHC, vHC, ’ob’); hold off
    ;
% Select the four corners of the A4 paper, in the order
%  up-left => up-right => bottom-right => bottom-left
% No need to click exactly on the four corners, just click
    nearby the
%  four corners and detected Harris corners are matched
    automatically
[u, v] = ginput(4);
for k=1:4 % match the corresponding Harris corners
    dist = abs(uHC-u(k))+abs(vHC-v(k));
    iMin = find(dist==min(dist)); iMin = iMin(1);
    u(k) = uHC(iMin); v(k) = vHC(iMin);
end
A4H = 0.297; A4W = 0.21; % A4 paper size is a priori known
Hm = CalibrateHomography(u, v, [0, A4W, A4W, 0], [0, 0, A4H, A4H
    ]);
[uProj, vProj] = HomographyXY2UV(Hm, [0, A4W, A4W, 0], [0, 0,
    A4H, A4H]);
figure(1), hold on; plot(uProj, vProj, ’+r’); hold off; pause(5)
    ;
% Generate the normal view image
sc = 0.0001; % pixel scale
H=round(A4H/sc); W=round(A4W/sc); imgs=uint8(zeros(H,W,size(imgo
    ,3)));
c = reshape(repmat([1:W],H,1),1,[]); xw = sc*c;
r = reshape(repmat([1:H]’,1,W),1,[]); yw = sc*r;
[u, v] = HomographyXY2UV(Hm, xw, yw); % projection to the camera
     image
for k=1:length(u)
    uf = floor(u(k)); vf = floor(v(k)); du = u(k)-uf; dv = v(k)-
    vf;
    % linear interpolation of the camera image pixel values
    imgs(r(k),c(k),:) = imgo(vf,uf,:)*(1-du)*(1-dv) ...
                        +imgo(vf,uf+1,:)*du*(1-dv) ...
                        +imgo(vf+1,uf,:)*(1-du)*dv ...
                        +imgo(vf+1,uf+1,:)*du*dv;
end
figure(2), imshow(imgs); imwrite(imgs,’
    tribute_to_yanan_NormalView.jpg’);
```

The involved sub-functions **HomographyXY2UV** and **CalibrateHomography** are given in previous sub-sections of Sect. 4.3. The involved sub-function **CornerHarris** is given in Sect. 2.4 in Chap. 2.

As demonstrated in the Matlab script **demoVirtualFileScanning.m**, the actual implementation of inverse perspective mapping does not follow the formalism of inverse perspective mapping literally. In other words, one does not really perform inverse projection of the original camera image pixels onto the target plane, i.e. the A4 paper plane.

Instead, one can generate the normal view image by projecting its pixels onto the original camera image and hence setting the corresponding values that its pixels should take. More specifically, consider the A4 paper area on the target plane, which is the rectangle area spanned by the four corners

$$(0.000, 0.000), \quad (0.210, 0.000), \quad (0.210, 0.297), \quad (0.000, 0.297).$$

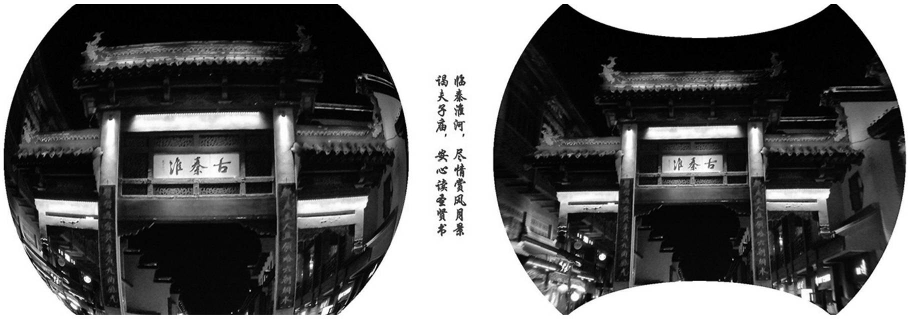
Discretize the rectangle area according to certain pixel scale—In **demoVirtualFileScanning.m**, the pixel scale is set to 0.0001, which means each pixel of the generated normal view image represents a $0.0001 \times 0.0001\mathrm{m}^2$ cell and hence the normal view image has a resolution of 2970×2100 pixels—For each pixel of the normal view image, one can know its corresponding world coordinates and compute its projection in the original camera image. Then obtain its value by linear interpolation of relevant pixel values of the original camera image.

Although the presented implementation does not follow the formalism of inverse perspective mapping literally, it indeed follows the spirit of inverse perspective mapping. Besides, compared with the implementation that follows the formalism of inverse perspective mapping literally, the presented implementation is better in the sense that it is more convenient[9] and it generates a smoother normal view image.[10]

4.4 Image Distortion

The *pinhole camera model* described in Sect. 4.2 is the ideal camera model also known as the *distortion-free camera model* . It is acceptable in practical applications when image distortion is negligible, which usually holds for conventional cameras, i.e. narrow-angle cameras.

In contrast, some practical applications necessitate usage of fish-eye cameras, i.e. wide-angle cameras [14–16] that incur unnegligible and even severe image distortion. In such applications, image distortion needs to be taken into account in the camera model.



(a) (b)

Fig. 4.12 Image distortion rectification **a** original camera image with distortion; **b** distortion-rectified image

An example of image distortion effect is illustrated in Fig. 4.12a, where one can see evident bending of straight lines towards the image central part (more specifically, towards the principal point). Besides, the further the pixels are away from the image central part, the more evident the bending is.

4.4.1 Distortion-Considered Camera Model

In 1906, the first fish-eye camera or wide-angle camera appeared [17], which was made by placing a normal camera lens in an apparatus filled with water, as illustrated in Fig. 4.13. It follows the physical phenomenon that from under the water one can see a wider view than what one can see directly in the air.

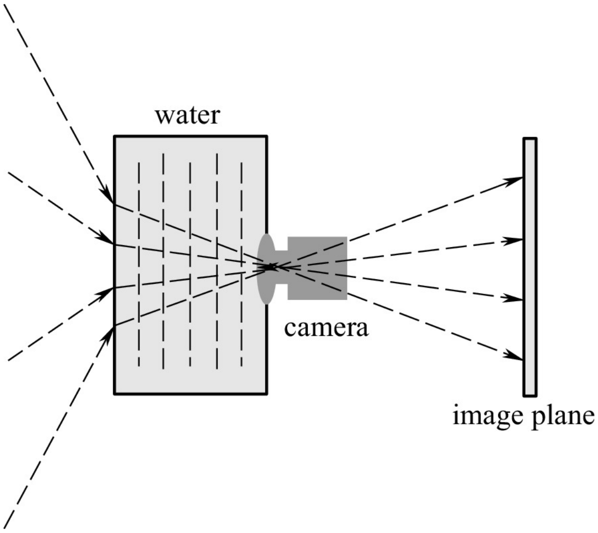


Fig. 4.13 First fish-eye camera: vision under water

Years latter after its birth, such kind of camera was improved by placing a normal camera lens in concatenation with a glass hemisphere [18], instead of placing it under the water. The working mechanism of the glass hemisphere concatenated wide-angle camera is illustrated in Fig. 4.14.

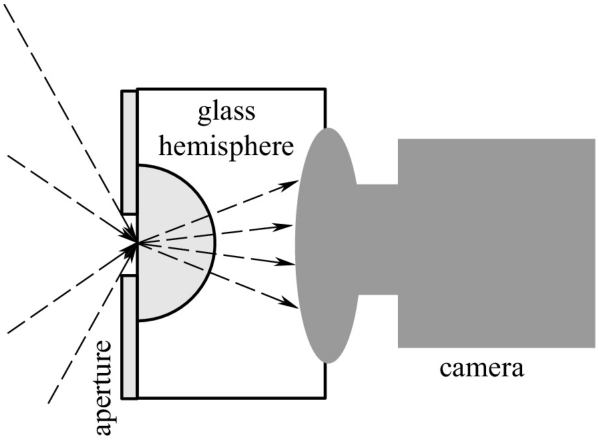


Fig. 4.14 Wide-angle camera: vision in concatenation with glass hemisphere

As demonstrated in Figs. 4.13 and 4.14, image distortion is essentially caused by light refraction that happens when light passes from one medium to another medium (i.e. from air to water and from air to glass in the two concrete cases respectively). Then by taking light refraction into account in the pinhole camera model illustrated in Fig. 4.7, one can establish the **distortion-considered camera model** , as illustrated in Fig. 4.15.

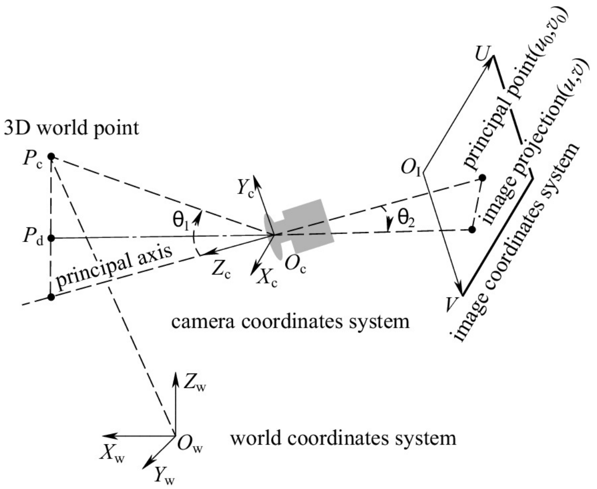


Fig. 4.15 Distortion-considered camera model: image distortion caused by light refraction

To further formalize the distortion-considered camera model , resort to a fundamental physical law in optics namely the *Fermat principle of refraction* [19]. The principle is the refraction case of the general *Fermat principle of least time*, which states *light* takes the path that requires the *least time* or *shortest time* to get from one point to another point, out of all possible paths that it might take to pass the two points.

The *Fermat principle of least time* implies the well-known formula of light refraction

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_1}{n_2} = n_{12}, \tag{4.26}$$

where θ_1 denotes the *angle of incidence* in the first medium, θ_2 denotes the *angle of refraction* in the second medium, n_1 denotes the *index* of the first medium against vacuum (namely the ratio of the light speed in the first medium with respect to that in vacuum), and n_2 denotes the *index* of the second medium against vacuum—the ratio n_1/n_2 is the same to n_{12}, i.e. the index of the first medium against the second medium.

Proof Suppose the plane that separates the two media is $z = 0$, two generic points in the two media are $(0, 0, h_1)^{\mathrm{T}}$ and $(L, 0, -h_2)^{\mathrm{T}}$ respectively, and the refracted point is $(x, y, 0)^{\mathrm{T}}$, then the time for light to pass the two points is a function in terms of (x, y):

$$t(x,y) \propto \frac{\sqrt{x^2+y^2+h_1^2}}{n_1} + \frac{\sqrt{(L-x)^2+y^2+h_2^2}}{n_2} \geqslant \frac{\sqrt{x^2+h_1^2}}{n_1} + \frac{\sqrt{(L-x)^2+h_2^2}}{n_2},$$

where the equality holds if and only if $y = 0$. Further consider the extremum condition and obtain

$$\begin{aligned} \frac{\partial}{\partial x} t(x,0) &= \frac{x}{n_1\sqrt{x^2+h_1^2}} - \frac{(L-x)}{n_2\sqrt{(L-x)^2+h_2^2}} \\ &= \frac{\sin\theta_1}{n_1} - \frac{\sin\theta_2}{n_2} = 0 \Longleftrightarrow \frac{\sin\theta_1}{\sin\theta_2} = \frac{n_1}{n_2}. \end{aligned}$$

Proof of the formula (4.26) is done.□

Given a generic 3D world point $\boldsymbol{P}_{\mathrm{c}} \equiv [x_{\mathrm{c}} \quad y_{\mathrm{c}} \quad z_{\mathrm{c}}]^{\mathrm{T}}$ expressed in the camera coordinates system. The incidence ray comes from the 3D world point and the refracted ray is projected to the image coordinates system. The angle of incidence is

$$\theta_1 = \arcsin \frac{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}}{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2+z_\mathrm{c}^2}}. \tag{4.27}$$

Via the formula of light refraction (4.26), one can derive the angle of refraction

$$\begin{aligned} \theta_2 &= \arcsin \frac{\sin\theta_1}{n_{12}} = \arcsin \frac{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}}{n_{12}\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2+z_\mathrm{c}^2}} \\ &= \arctan \frac{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}}{\sqrt{(n_{12}^2-1)(x_\mathrm{c}^2+y_\mathrm{c}^2)+n_{12}^2 z_\mathrm{c}^2}} = \arctan \frac{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}}{n_{12} z_\mathrm{c} \sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2}}}, \end{aligned} \tag{4.28}$$

where

$n_{12}^* \equiv 1 - \frac{1}{n_{12}^2} > 0$ as $n_{12} > 1$.

The refracted ray seems to be emitted from a virtual 3D world point $\boldsymbol{P}_\mathrm{d} \equiv [x_\mathrm{d} \quad y_\mathrm{d} \quad z_\mathrm{d}]^\mathrm{T}$ which satisfies

$$x_\mathrm{d} = \frac{x_\mathrm{c}}{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}} z_\mathrm{c} \tan\theta_2 = \frac{x_\mathrm{c}}{n_{12}\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2}}}, \tag{4.29a}$$

$$y_\mathrm{d} = \frac{y_\mathrm{c}}{\sqrt{x_\mathrm{c}^2+y_\mathrm{c}^2}} z_\mathrm{c} \tan\theta_2 = \frac{y_\mathrm{c}}{n_{12}\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2}}}, \tag{4.29b}$$

$$z_\mathrm{d} = z_\mathrm{c}. \tag{4.29c}$$

The physical meaning of (4.29) is interpreted as follows: the virtual 3D world point $\boldsymbol{P}_\mathrm{d}$ is obtained from the actual 3D world point $\boldsymbol{P}_\mathrm{c}$ by contracting the coordinates $(x_\mathrm{c}, y_\mathrm{c})$ along the *radial* direction from $[x_\mathrm{c} \quad y_\mathrm{c} \quad z_\mathrm{c}]^\mathrm{T}$ towards $[0 \quad 0 \quad z_\mathrm{c}]^\mathrm{T}$, and the contraction factor depends on the index of the first medium against the second medium and the ratio of the radial offset $\sqrt{x_\mathrm{c}^2 + y_\mathrm{c}^2}$ with respect to the viewing distance z_c. Such effect of virtual contraction is illustrated in Fig. 4.15.

Similar to (4.2), the transform between the virtual camera coordinates system (namely one that expresses the point $\boldsymbol{P}_\mathrm{d}$ virtually contracted from the actual 3D world point $\boldsymbol{P}_\mathrm{c}$) and the image coordinates system is formalized as

$$u = \alpha^* \frac{x_\mathrm{d}}{z_\mathrm{d}} + \gamma^* \frac{y_\mathrm{d}}{z_\mathrm{d}} + u_0, \tag{4.30a}$$

$$v = \beta^* \frac{y_\mathrm{d}}{z_\mathrm{d}} + v_0. \tag{4.30b}$$

Substitute (4.29) into (4.30) and obtain

$$u = \left(\alpha \frac{x_\mathrm{c}}{z_\mathrm{c}} + \gamma \frac{y_\mathrm{c}}{z_\mathrm{c}}\right) \frac{1}{\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2}}} + u_0, \tag{4.31a}$$

$$v \quad = \beta \tfrac{y_\mathrm{c}}{z_\mathrm{c}} \frac{1}{\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2}}} + v_0, \tag{4.31b}$$

where

$$\alpha \equiv \tfrac{\alpha^*}{n_{12}}, \quad \gamma \equiv \tfrac{\gamma^*}{n_{12}}, \quad \beta \equiv \tfrac{\beta^*}{n_{12}}.$$

Denote the ideal (distortion-free) image coordinates [11] $(\hat{u}, \hat{v})$ as

$$\hat{u} \quad = \alpha \tfrac{x_\mathrm{c}}{z_\mathrm{c}} + \gamma \tfrac{y_\mathrm{c}}{z_\mathrm{c}} + u_0, \tag{4.32a}$$

$$\hat{v} \quad = \beta \tfrac{y_\mathrm{c}}{z_\mathrm{c}} + v_0 \tag{4.32b}$$

and express

$$\begin{aligned} \alpha \tfrac{x_\mathrm{c}}{z_\mathrm{c}} + \gamma \tfrac{y_\mathrm{c}}{z_\mathrm{c}} &= \hat{u} - u_0, \\ \beta \tfrac{y_\mathrm{c}}{z_\mathrm{c}} &= \hat{v} - v_0. \end{aligned}$$

Substitute (4.32) into (4.31) and obtain

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \frac{1}{\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2}}} \tag{4.33}$$

Note the power series

$$\tfrac{1}{\sqrt{1+x}} = 1 - \tfrac{1}{2}x + \tfrac{3}{8}x^2 - \tfrac{5}{16}x^3 + \cdots$$

So the factor $1/\sqrt{1 + n_{12}^* \frac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2}}$ in (4.33) can be expanded into the power series form

$$\begin{aligned} &\frac{1}{\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2}}} \\ = \quad & 1 - \tfrac{n_{12}^*}{2} \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right) + \tfrac{3n_{12}^{*2}}{8} \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right)^2 - \tfrac{5n_{12}^{*3}}{16} \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right)^3 + \cdots \end{aligned} \tag{4.34}$$

and can be further formalized with generically-expressed coefficients

$$\frac{1}{\sqrt{1+n_{12}^* \dfrac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2}}} = 1 + d_1 \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right) + d_2 \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right)^2 + d_3 \left(\tfrac{x_\mathrm{c}^2+y_\mathrm{c}^2}{z_\mathrm{c}^2} \right)^3 + \cdots \tag{4.35}$$

The coefficients d_1, d_2, d_3, ... in (4.35) are theoretically correlated because they stem from the common coefficient n_{12}^* in (4.34). Yet in practice, treating the coefficients d_1, d_2, d_3, ...

as free coefficients enables flexible modelling of image distortion and facilitates theoretical analysis and numeric computation as well.[12]

Substitute (4.35) into (4.33) and obtain the *image distortion model*

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \left[1 + d_1 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right) + d_2 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right)^2 + \cdots \right], \tag{4.36}$$

where d_1, d_2, ... denote *distortion parameters*. The factor

$$\begin{aligned} s_{\boldsymbol{P}_{\mathrm{c}}} &= 1 + d_1 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right) + d_2 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right)^2 + \cdots \\ &= 1 + d_1 \left(\boldsymbol{P}_{\mathrm{c}}^{\mathrm{T}} \frac{\boldsymbol{e}_1 \boldsymbol{e}_1^{\mathrm{T}} + \boldsymbol{e}_2 \boldsymbol{e}_2^{\mathrm{T}}}{\boldsymbol{P}_{\mathrm{c}}^{\mathrm{T}} \boldsymbol{e}_3 \boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{P}_{\mathrm{c}}} \boldsymbol{P}_{\mathrm{c}} \right) + d_2 \left(\boldsymbol{P}_{\mathrm{c}}^{\mathrm{T}} \frac{\boldsymbol{e}_1 \boldsymbol{e}_1^{\mathrm{T}} + \boldsymbol{e}_2 \boldsymbol{e}_2^{\mathrm{T}}}{\boldsymbol{P}_{\mathrm{c}}^{\mathrm{T}} \boldsymbol{e}_3 \boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{P}_{\mathrm{c}}} \boldsymbol{P}_{\mathrm{c}} \right)^2 + \cdots \end{aligned} \tag{4.37}$$

in (4.36) is called the *image distortion scaling factor*. The image distortion model (4.36) can be formalized compactly as

$$\boldsymbol{P}_{\mathrm{i}} = \boldsymbol{P}_{\mathrm{i},0} + (\hat{\boldsymbol{P}}_{\mathrm{i}} - \mathbf{P}_{\mathrm{i},0}) s_{\mathbf{P}_{\mathrm{c}}},$$

where

$$\boldsymbol{P}_{\mathrm{i}} \equiv \begin{bmatrix} u \\ v \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},0} \equiv \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}, \quad \hat{\boldsymbol{P}}_{\mathrm{i}} \equiv \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{c}} \equiv \begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{bmatrix}.$$

The image distortion model (4.36) is also called the *image radial distortion model*, as image distortion exists along the radial direction from the principal point .

By combining (4.4), (4.6), and (4.36), one can obtain a complete formalism of the **distortion-considered camera model**

$$\begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{bmatrix} = [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix}, \tag{4.38a}$$

$$\lambda \begin{bmatrix} \hat{u} \\ \hat{v} \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{bmatrix}, \tag{4.38b}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \left[1 + d_1 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right) + d_2 \left(\frac{x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} \right)^2 + \cdots \right] \tag{4.38c}$$

or in the compact form

$$\begin{aligned}
\boldsymbol{P}_{\mathrm{c}} &= [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w}}^{\mathrm{A}}, \\
\lambda\hat{\boldsymbol{P}}_{\mathrm{i}}^{\mathrm{A}} &= \boldsymbol{A}\boldsymbol{P}_{\mathrm{c}}, \\
s_{\boldsymbol{P}_{\mathrm{c}}} &= 1 + d_1\left(\boldsymbol{P}_{\mathrm{c}}{}^{\mathrm{T}}\frac{\mathbf{e}_1\mathbf{e}_1^{\mathrm{T}}+\mathbf{e}_2\mathbf{e}_2^{\mathrm{T}}}{\mathbf{P}_{\mathrm{c}}^{\mathrm{T}}\mathbf{e}_3\mathbf{e}_3^{\mathrm{T}}\mathbf{P}_{\mathrm{c}}}\mathbf{P}_{\mathrm{c}}\right) + \mathrm{d}_2\left(\mathbf{P}_{\mathrm{c}}^{\mathrm{T}}\frac{\mathbf{e}_1\mathbf{e}_1^{\mathrm{T}}+\mathbf{e}_2\mathbf{e}_2^{\mathrm{T}}}{\mathbf{P}_{\mathrm{c}}^{\mathrm{T}}\mathbf{e}_3\mathbf{e}_3^{\mathrm{T}}\mathbf{P}_{\mathrm{c}}}\mathbf{P}_{\mathrm{c}}\right)^2 + \cdots \\
\boldsymbol{P}_{\mathrm{i}} &= \boldsymbol{P}_{\mathrm{i},0} + (\hat{\boldsymbol{P}}_{\mathrm{i}} - \mathbf{P}_{\mathrm{i},0})\mathrm{s}_{\mathbf{P}_{\mathrm{c}}}.
\end{aligned}$$

The first and second equations of (4.38) are essentially the same to the formalism of the pinhole camera model presented in Sect. 4.2. The third equation of (4.38) is the formalism of the image distortion model just presented.

Variants of the image distortion model

It is worth noting that different image distortion models (or distortion models for short) can exist, depending on how various physical factors that cause image distortion are taken into account [20, 21]. For example, besides distortion caused by standard light refraction, distortion caused by other physical factors such as “decentering distortion” and “thin prism distortion” may also be taken into account[21]. However, the author neglects such kinds of distortion, because they are not so general in practice and sometimes their influence can indeed be neglected. The author focuses on distortion caused only by standard light refraction and have derived accordingly the distortion model formalized in (4.36) by so far.

However, the distortion model formalized in (4.36) involves camera coordinates $(x_{\mathrm{c}}, y_{\mathrm{c}}, z_{\mathrm{c}})$ explicitly. If one intends to analyse and process image distortion only with image coordinates without entangling the camera coordinates system, then (4.36) turns to be inappropriate. For sake of analysing and processing image distortion only with image coordinates, one needs to develop other variants of the distortion model.

According to (4.32), one can express $x_{\mathrm{c}}/z_{\mathrm{c}}$, $y_{\mathrm{c}}/z_{\mathrm{c}}$, and $(x_{\mathrm{c}}^2 + y_{\mathrm{c}}^2)/z_{\mathrm{c}}^2$ in terms of $\hat{u}$ and $\hat{v}$:

$$\frac{x_{\mathrm{c}}}{z_{\mathrm{c}}} = \frac{\hat{u}-u_0}{\alpha} - \frac{\gamma(\hat{v}-v_0)}{\alpha\beta} \equiv \frac{\Delta\hat{u}}{\alpha} - \frac{\gamma\Delta\hat{v}}{\alpha\beta}, \tag{4.39a}$$

$$\frac{y_{\mathrm{c}}}{z_{\mathrm{c}}} = \frac{\hat{v}-v_0}{\beta} \equiv \frac{\Delta\hat{v}}{\beta}, \tag{4.39b}$$

$$\frac{x_{\mathrm{c}}^2+y_{\mathrm{c}}^2}{z_{\mathrm{c}}^2} = \left(\frac{\Delta\hat{u}}{\alpha} - \frac{\gamma\Delta\hat{v}}{\alpha\beta}\right)^2 + \left(\frac{\Delta\hat{v}}{\beta}\right)^2 = \frac{1}{\alpha^2}\left(\Delta\hat{u}^2 - 2\frac{\gamma}{\beta}\Delta\hat{u}\Delta\hat{v} + \frac{\alpha^2+\gamma^2}{\beta^2}\Delta\hat{v}^2\right), \tag{4.39c}$$

where

$$\Delta\hat{u} \equiv \hat{u} - u_0, \quad \Delta\hat{v} \equiv \hat{v} - v_0.$$

Substitute (4.32) and (4.39) into (4.36) and formalize (4.36) equivalently as

$$\begin{aligned}
\begin{bmatrix} u \\ v \end{bmatrix} = \quad & \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix}\Bigg[1 + \frac{d_1}{\alpha^2}\left(\Delta\hat{u}^2 - 2\frac{\gamma}{\beta}\Delta\hat{u}\Delta\hat{v} + \frac{\alpha^2+\gamma^2}{\beta^2}\Delta\hat{v}^2\right) + \\
& \frac{d_2}{\alpha^4}\left(\Delta\hat{u}^2 - 2\frac{\gamma}{\beta}\Delta\hat{u}\Delta\hat{v} + \frac{\alpha^2+\gamma^2}{\beta^2}\Delta\hat{v}^2\right)^2 + \cdots\Bigg],
\end{aligned} \tag{4.40}$$

where the newly formalized image distortion scaling factor

$$s_{\hat{\boldsymbol{P}}_{\mathrm{i}}} = \begin{aligned} & 1 + \tfrac{d_1}{\alpha^2}\left(\Delta\hat{u}^2 - 2\tfrac{\gamma}{\beta}\Delta\hat{u}\Delta\hat{v} + \tfrac{\alpha^2+\gamma^2}{\beta^2}\Delta\hat{v}^2\right) + \\ & \tfrac{d_2}{\alpha^4}\left(\Delta\hat{u}^2 - 2\tfrac{\gamma}{\beta}\Delta\hat{u}\Delta\hat{v} + \tfrac{\alpha^2+\gamma^2}{\beta^2}\Delta\hat{v}^2\right)^2 + \cdots \end{aligned} \tag{4.41}$$

is expressed only in terms of image coordinates without entangling the camera coordinates system. Expand (4.41) and express the image distortion scaling factor $s_{\hat{\boldsymbol{P}}_{\mathrm{i}}}$ as summation of groups of homogeneous terms with generically denoted coefficients

$$s_{\hat{\boldsymbol{P}}_{\mathrm{i}}} = 1 + \sum_{i+j=2} k_{1,i,j}\Delta\hat{u}^i\Delta\hat{v}^j + \sum_{i+j=4} k_{2,i,j}\Delta\hat{u}^i\Delta\hat{v}^j + \sum_{i+j=6} k_{3,i,j}\Delta\hat{u}^i\Delta\hat{v}^j + \cdots \tag{4.42}$$

where all subscripts i and j in the summation terms are non-negative.

The completely-parametrized formalism described in (4.42) is hardly used in practical applications. Note that in practice one usually has

$$\alpha \approx \beta, \quad \gamma \ll \alpha, \beta,$$

so the formalism of the image distortion scaling factor $s_{\hat{\boldsymbol{P}}_{\mathrm{i}}}$ in (4.41) is simplified to

$$s_{\hat{\boldsymbol{P}}_{\mathrm{i}}} = 1 + \tfrac{d_1}{\alpha^2}(\Delta\hat{u}^2 + \Delta\hat{v}^2) + \tfrac{d_2}{\alpha^4}(\Delta\hat{u}^2 + \Delta\hat{v}^2)^2 + \cdots \tag{4.43}$$

or

$$\begin{aligned} s_{\hat{\boldsymbol{P}}_{\mathrm{i}}} &= 1 + k_1(\Delta\hat{u}^2 + \Delta\hat{v}^2) + k_2(\Delta\hat{u}^2 + \Delta\hat{v}^2)^2 + \cdots \\ &= 1 + k_1\left\|\begin{bmatrix}\hat{u} - u_0 \\ \hat{v} - v_0\end{bmatrix}\right\|_2^2 + k_2\left\|\begin{bmatrix}\hat{u} - u_0 \\ \hat{v} - v_0\end{bmatrix}\right\|_2^4 + \cdots \\ &= 1 + k_1\|\hat{\boldsymbol{P}}_{\mathrm{i}} - \boldsymbol{P}_{\mathrm{i},0}\|_2^2 + k_2\|\hat{\boldsymbol{P}}_{\mathrm{i}} - \boldsymbol{P}_{\mathrm{i},0}\|_2^4 + \cdots \end{aligned} \tag{4.44}$$

where $k_1 = d_1/\alpha^2$, $k_2 = d_2/\alpha^4$, ... also denote distortion parameters. Substitute (4.44) into (4.40) and obtain a new variant of image distortion model

$$\begin{bmatrix}u \\ v\end{bmatrix} = \begin{bmatrix}u_0 \\ v_0\end{bmatrix} + \begin{bmatrix}\hat{u} - u_0 \\ \hat{v} - v_0\end{bmatrix}\left(1 + k_1\left\|\begin{bmatrix}\hat{u} - u_0 \\ \hat{v} - v_0\end{bmatrix}\right\|_2^2 + k_2\left\|\begin{bmatrix}\hat{u} - u_0 \\ \hat{v} - v_0\end{bmatrix}\right\|_2^4 + \cdots\right). \tag{4.45}$$

By combining (4.7) and (4.45), one can obtain a variant of the distortion-considered camera model

$$\lambda\begin{bmatrix}\hat{u} \\ \hat{v} \\ 1\end{bmatrix} = \begin{bmatrix}\alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{R}_{\mathrm{wc}} & \boldsymbol{T}_{\mathrm{wc}}\end{bmatrix}\begin{bmatrix}x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1\end{bmatrix}, \tag{4.46a}$$

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}u_0\\v_0\end{bmatrix}+\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\left(1+k_1\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2^2+k_2\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2^4+\cdots\right) \tag{4.46b}$$

or in the compact form

$$\begin{aligned}\lambda\hat{\boldsymbol{P}}_\mathrm{i}^\mathrm{A} &= \boldsymbol{A}[\boldsymbol{R}_\mathrm{wc} \quad \boldsymbol{T}_\mathrm{wc}]\boldsymbol{P}_\mathrm{w}^\mathrm{A},\\ s_{\hat{\boldsymbol{P}}_\mathrm{i}} &= 1+k_1\|\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0}\|_2^2+k_2\|\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0}\|_2^4+\cdots\\ \boldsymbol{P}_\mathrm{i} &= \boldsymbol{P}_\mathrm{i,0}+(\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0})s_{\hat{\boldsymbol{P}}_\mathrm{i}}.\end{aligned}$$

The scale of the image may sometimes be adjusted, which will result in a change of distortion parameters in (4.45). To maintain distortion parameter consistency despite potential image scaling, one may transform (4.45) into

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}u_0\\v_0\end{bmatrix}+\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\left[1+k_1\left(\frac{\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2}{S_I}\right)^2+k_2\left(\frac{\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2}{S_I}\right)^4+\cdots\right] \tag{4.47}$$

where S_I represents the image scale and the distortion scaling factor is computed with normalized image coordinates. By the way, the author abuses notations k_1, k_2, ... in (4.47) as in (4.45) to avoid dazzling of new notations that bear no conceptually new sense other than distortion parameters.

By combining (4.7) and (4.47), one can obtain another variant of the distortion-considered camera model

$$\lambda\begin{bmatrix}\hat{u}\\\hat{v}\\1\end{bmatrix}=\begin{bmatrix}\alpha & \gamma & u_0\\0 & \beta & v_0\\0 & 0 & 1\end{bmatrix}[\boldsymbol{R}_\mathrm{wc} \quad \boldsymbol{T}_\mathrm{wc}]\begin{bmatrix}x_\mathrm{w}\\y_\mathrm{w}\\z_\mathrm{w}\\1\end{bmatrix}, \tag{4.48a}$$

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}u_0\\v_0\end{bmatrix}+\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\left[1+k_1\left(\frac{\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2}{S_\mathrm{I}}\right)^2+k_2\left(\frac{\left\|\begin{bmatrix}\hat{u}-u_0\\\hat{v}-v_0\end{bmatrix}\right\|_2}{S_\mathrm{I}}\right)^4+\cdots\right] \tag{4.48b}$$

or in the compact form

$$\begin{aligned}\lambda\hat{\boldsymbol{P}}_\mathrm{i}^\mathrm{A} &= \boldsymbol{A}[\boldsymbol{R}_\mathrm{wc} \quad \boldsymbol{T}_\mathrm{wc}]\boldsymbol{P}_\mathrm{w}^\mathrm{A},\\ s_{\hat{\boldsymbol{P}}_\mathrm{i}} &= 1+k_1\left(\frac{\|\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0}\|_2}{S_I}\right)^2+k_2\left(\frac{\|\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0}\|_2}{S_\mathrm{I}}\right)^4+\cdots\\ \boldsymbol{P}_\mathrm{i} &= \boldsymbol{P}_\mathrm{i,0}+(\hat{\boldsymbol{P}}_\mathrm{i}-\boldsymbol{P}_\mathrm{i,0})s_{\hat{\boldsymbol{P}}_\mathrm{i}}.\end{aligned}$$

4.4.2 Inverse Distortion Model and Distortion Rectification

Although the distortion-considered camera model described by (4.38), (4.46), or (4.48) establishes explicit relationship between a generic 3D world point and its image projection, it is not always desirable to use the distortion-considered camera model directly in practical applications, because many useful aspects of invariance in projective geometry are nullified by image distortion and the nonlinear factors in the formalism (4.38), (4.46), or (4.48) incur inconvenience in terms of mathematical analysis.

It is hoped sometimes that in practical applications involving image distortion, the relationship between the 3D world environment and the image would still be handled as if the pinhole camera model is applicable. For this sake, a natural idea is to recover the distortion-free image coordinates $(\hat{u}, \hat{v})$ that exist virtually in one's imagination from the actual image coordinates (u, v).

Recovery of the distortion-free image coordinates $(\hat{u}, \hat{v})$ from the actual image coordinates (u, v) is called **distortion rectification**. Once distortion rectification is done, one obtains a rectified image which is exempt from image distortion, and the relationship between the 3D world environment and the rectified image is rightdescribed by the pinhole camera model .

Then a question arises naturally: how to recover the ideal (distortion-free) image coordinates $(\hat{u}, \hat{v})$ from the actual image coordinates (u, v)? Consider (4.45) and transform it into

$$\begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} = \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \left(1 + k_1 \left\| \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \right\|_2^2 + k_2 \left\| \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \right\|_2^4 + \cdots \right) \tag{4.49}$$

Compute the L_2-norms of both sides of (4.49) and obtain

$$r = \hat{r}(1 + k_1 \hat{r}^2 + k_2 \hat{r}^4 + \cdots) \equiv \hat{r} s(\hat{r}), \tag{4.50}$$

where

$$\hat{r} = \left\| \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \right\|_2, \quad r = \left\| \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \right\|_2, \quad s(\hat{r}) = 1 + k_1 \hat{r}^2 + k_2 \hat{r}^4 + \cdots \tag{4.51}$$

Then (4.49) is formalized shortly as

$$\begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} = \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} s(\hat{r}). \tag{4.52}$$

Apply the *inverse function theorem*[13] to (4.50), there exists certain unique function

$$h(r) = h_1 r + h_2 r^2 + h_3 r^3 + h_4 r^4 + h_5 r^5 + \cdots$$

such that

$$\begin{aligned} \hat{r} = \quad & h(\hat{r} s(\hat{r})) = h(r) = h_1 r + h_2 r^2 + h_3 r^3 + h_4 r^4 + h_5 r^5 + \cdots \\ = \quad & h_1(\hat{r} + k_1 \hat{r}^3 + k_2 \hat{r}^5 + \cdots) + h_2 (\hat{r} + k_1 \hat{r}^3 + k_2 \hat{r}^5 + \cdots)^2 \\ & + h_3 (\hat{r} + k_1 \hat{r}^3 + k_2 \hat{r}^5 + \cdots)^3 + h_4 (\hat{r} + k_1 \hat{r}^3 + k_2 \hat{r}^5 + \cdots)^4 + \cdots \end{aligned} \tag{4.53}$$

Compare corresponding terms on both sides of (4.53): It is evident that $h_1 = 1$. Besides, note that the right-hand-side terms associated with h_1, h_3, h_5, ... provide $\hat{r}$ powers of odd orders, whereas the right-hand-side terms associated with h_2, h_4, h_6, ... provide $\hat{r}$ powers of even orders, and also note that the left-hand side has a single term $\hat{r}$ without $\hat{r}$ powers of even orders. So one can derive recursively

$$h_2 = 0, \quad h_4 = 0, \quad h_6 = 0, \quad \cdots$$

namely all h coefficients with even subscripts are zero.

So $\hat{r}$ expressed of r is of the form

$$\hat{r} = r + k_1^* r^3 + k_2^* r^5 + k_3^* r^7 + \cdots \equiv r s^*(r), \tag{4.54}$$

where

$$s^*(r) = 1 + k_1^* r^2 + k_2^* r^4 + k_3^* r^6 + \cdots$$

From (4.50), (4.52), and (4.54) one can know

$$\begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \frac{1}{s(\hat{r})} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \frac{1}{r/\hat{r}} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \frac{\hat{r}}{r} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} s^*(r).$$

Transform above equation equivalently into

$$\begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \left(1 + k_1^* \left\| \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \right\|_2^2 + k_2^* \left\| \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \right\|_2^4 + \cdots \right) \tag{4.55}$$

which is called the *inverse image distortion model*. It is worth noting the formalism duality between (4.45) and (4.55).

Once image distortion coefficients are calibrated (see Chap. 5), distortionrectification can be done[14] to obtain rectified images exempt from image distortion. Therefore, throughout the book, it is assumed by default that images involved in practical applications are ideal or distortion-free images and the pinhole camera model can always be applied.

4.5 Stereo Vision

If a vision system consists of at least two cameras, then it provides an ability called **stereo vision** (i.e. “natural” vision as if in the real world). Such a vision system is called a **stereo vision system** or simply a **stereo camera**. The most representative form of stereo vision is **binocular vision** (i.e. “two-eyes” or “two-cameras” vision),[15] so without explicit mentioning, stereo vision in following presentation means binocular vision by default, and a stereo vision system or a stereo camera means a binocular vision system by default. After all, knowledge on binocular vision can be naturally extended to stereo vision that involves more cameras.

4.5.1 Why Stereo Vision Is “Natural”?

Stereo vision is “natural” in the sense that it enables localization of geometric entities in the three-dimensional space of the real world and hence gives one a true feeling of existence in the real world. In contrast, monocular vision lacks such naturalness for its incapability of localizing geometric entities in the three-dimensional space. Why do stereo vision and monocular vision have such essential difference?

Consider the pinhole camera model described by (4.7)

$$\lambda \boldsymbol{P}_{\mathrm{i}}^{\mathrm{A}} \equiv \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} \equiv \boldsymbol{A}[\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w}}^{\mathrm{A}}$$

or in the perspective mapping form as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{M}_{1,:} \\ \boldsymbol{M}_{2,:} \\ \boldsymbol{M}_{3,:} \end{bmatrix} \boldsymbol{P}_{\mathrm{w}}^{\mathrm{A}}.$$

Given a known image point (u, v), the formalism (4.7) conveys two equations in terms of the world coordinates x_{w}, y_{w}, and z_{w} as

$$(m_{11} - u \cdot m_{31})x_{\mathrm{w}} + (m_{12} - u \cdot m_{32})y_{\mathrm{w}} + (m_{13} - u \cdot m_{33})z_{\mathrm{w}} + m_{14} - u \cdot m_{34} = 0, \tag{4.56a}$$

$$(m_{21} - v \cdot m_{31})x_{\mathrm{w}} + (m_{22} - v \cdot m_{32})y_{\mathrm{w}} + (m_{23} - v \cdot m_{33})z_{\mathrm{w}} + m_{24} - v \cdot m_{34} = 0. \tag{4.56b}$$

or compactly as

$$\begin{bmatrix} \boldsymbol{M}_{1,:} - u \cdot \boldsymbol{M}_{3,:} \\ \boldsymbol{M}_{2,:} - v \cdot \boldsymbol{M}_{3,:} \end{bmatrix} \boldsymbol{P}_{\mathrm{w}}^{A} = \boldsymbol{0}.$$

Two equations are insufficient to determine three unknowns x_{w}, y_{w}, and z_{w}. This is why monocular vision is incapable of three-dimensional localization.

$$\begin{aligned} \lambda^{[k]} \boldsymbol{P}_{\mathrm{i}}^{[\mathrm{k}]\mathrm{A}} &\equiv \lambda \begin{bmatrix} u^{[k]} \\ v^{[k]} \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha^{[k]} & \gamma^{[k]} & u_0^{[k]} \\ 0 & \beta^{[k]} & v_0^{[k]} \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}}^{[k]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} \\ &\equiv \boldsymbol{A}^{[k]}[\boldsymbol{R}_{\mathrm{wc}}^{[k]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}]\boldsymbol{P}_{\mathrm{w}}^{\mathrm{A}} \end{aligned} \tag{4.57}$$

or in the perspective mapping form as

$$\lambda^{[k]} \begin{bmatrix} u^{[k]} \\ v^{[k]} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}^{[k]} & m_{12}^{[k]} & m_{13}^{[k]} & m_{14}^{[k]} \\ [6pt]m_{21}^{[k]} & m_{22}^{[k]} & m_{23}^{[k]} & m_{24}^{[k]} \\ [6pt]m_{31}^{[k]} & m_{32}^{[k]} & m_{33}^{[k]} & m_{34}^{[k]} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{M}_{1,:}^{[k]} \\ [6pt]\boldsymbol{M}_{2,:}^{[k]} \\ [6pt]\boldsymbol{M}_{3,:}^{[k]} \end{bmatrix} \boldsymbol{P}_{\mathrm{w}}^{A},$$

where $k \in \{1, 2\}$ denotes the camera index.

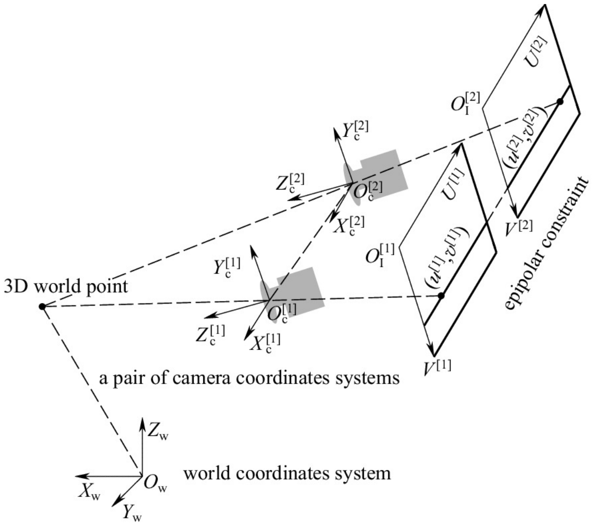

$U^{[2]}$
$O_{\rm I}^{[2]}$
$(u^{[2]},v^{[2]})$
$Y_{\rm c}^{[2]}$
$Z_{\rm c}^{[2]}$
$O_{\rm c}^{[2]}$
$X_{\rm c}^{[2]}$
$U^{[1]}$
$V^{[2]}$
$O_{\rm I}^{[1]}$
$(u^{[1]},v^{[1]})$
epipolar constraint
$Y_{\rm c}^{[1]}$
3D world point
$O_{\rm c}^{[1]}$
$Z_{\rm c}^{[1]}$
$X_{\rm c}^{[1]}$
$V^{[1]}$
a pair of camera coordinates systems
$Z_{\rm w}$
$X_{\rm w}$
$O_{\rm w}$
world coordinates system
$Y_{\rm w}$

Fig. 4.16 Stereo vision: epipolar constraint

Given a known pair of image points $(u^{[1]}, v^{[1]})$ and $(u^{[2]}, v^{[2]})$, the formalism (4.57) conveys four equations in terms of the world coordinates $x_{\rm w}$, $y_{\rm w}$, and $z_{\rm w}$ as

$$(m_{11}^{[1]}-u^{[1]}m_{31}^{[1]})x_{\rm w}+(m_{12}^{[1]}-u^{[1]}m_{32}^{[1]})y_{\rm w}+(m_{13}^{[1]}-u^{[1]}m_{33}^{[1]})z_{\rm w}+m_{14}^{[1]}-u^{[1]}m_{34}^{[1]} = 0, \tag{4.58a}$$

$$(m_{21}^{[1]}-v^{[1]}m_{31}^{[1]})x_{\rm w}+(m_{22}^{[1]}-v^{[1]}m_{32}^{[1]})y_{\rm w}+(m_{23}^{[1]}-v^{[1]}m_{33}^{[1]})z_{\rm w}+m_{24}^{[1]}-v^{[1]}m_{34}^{[1]} = 0, \tag{4.58b}$$

$$(m_{11}^{[2]}-u^{[2]}m_{31}^{[2]})x_{\rm w}+(m_{12}^{[2]}-u^{[2]}m_{32}^{[2]})y_{\rm w}+(m_{13}^{[2]}-u^{[2]}m_{33}^{[2]})z_{\rm w}+m_{14}^{[2]}-u^{[2]}m_{34}^{[2]} = 0, \tag{4.58c}$$

$$(m_{21}^{[2]}-v^{[2]}m_{31}^{[2]})x_{\rm w}+(m_{22}^{[2]}-v^{[2]}m_{32}^{[2]})y_{\rm w}+(m_{23}^{[2]}-v^{[2]}m_{33}^{[2]})z_{\rm w}+m_{24}^{[2]}-v^{[2]}m_{34}^{[2]} = 0. \tag{4.58d}$$

or compactly as

$$\begin{bmatrix} \boldsymbol{M}_{1,:}^{[1]} - u^{[1]}\boldsymbol{M}_{3,:}^{[1]} \\ \lbrack 6pt\rbrack \boldsymbol{M}_{2,:}^{[1]} - v^{[1]}\boldsymbol{M}_{3,:}^{[1]} \\ \lbrack 6pt\rbrack \boldsymbol{M}_{1,:}^{[2]} - u^{[2]}\boldsymbol{M}_{3,:}^{[2]} \\ \lbrack 6pt\rbrack \boldsymbol{M}_{2,:}^{[2]} - v^{[2]}\boldsymbol{M}_{3,:}^{[2]} \end{bmatrix} \boldsymbol{P}_{\rm w}^{A} = \boldsymbol{0}.$$

Four equations are sufficient to determine three unknowns $x_{\rm w}$, $y_{\rm w}$, and $z_{\rm w}$—degenerate cases, which are hardly encountered in practical applications of stereo vision, are neglected here—This is why stereo vision enables localization of geometric entities in the three-dimensional real world and gives one a feeling a naturalness.

4.5.2 Epipolar Constraint: Fundamental and Essential Matrices

There are four equations in (4.58) but there are only three unknowns $x_{\rm w}$, $y_{\rm w}$, and $z_{\rm w}$ to be determined, so the four equations in (4.58) are not independent. This further implies that the pair of image coordinates $\{(u^{[1]}, v^{[1]});(u^{[2]}, v^{[2]})\}$ are not independent either; otherwise, the four equations in (4.58) would be independent and cause contradiction. There must be certain geometric constraint that correlates image coordinates $\{(u^{[1]}, v^{[1]}); (u^{[2]}, v^{[2]})\}$ with each other. Figure 4.16 gives an intuitive illustration of such geometric constraint, which is called the **epipolar constraint** [23].

Besides geometric illustration, analytical clarification of the epipolar constraint is given below. Consider (4.57), treat $\lambda^{[1]}$ and $\lambda^{[2]}$ as two non-degenerate parameters, and express the world coordinates $\boldsymbol{P}_{\mathrm{w}}$ as

$$\boldsymbol{P}_{\mathrm{w}} = \boldsymbol{R}_{\mathrm{wc}}^{[k]\mathrm{T}}[\lambda^{[k]}(\boldsymbol{A}^{[k]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[k]A} - \boldsymbol{T}_{\mathrm{wc}}^{[k]}] = \lambda^{[k]}\boldsymbol{R}_{\mathrm{wc}}^{[k]\mathrm{T}}(\boldsymbol{A}^{[k]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[k]A} - \boldsymbol{R}_{\mathrm{wc}}^{[k]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[k]}, \tag{4.59}$$

where $\boldsymbol{T}_{\mathrm{cw}}^{[k]} = -\boldsymbol{R}_{\mathrm{wc}}^{[k]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[k]}$ denotes the camera coordinates system origin expressed in the world coordinates system. Equation (4.59) implies an equality

$$\begin{aligned} & \lambda^{[1]}\boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} - \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[1]} = \lambda^{[2]}\boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} - \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[2]} \\ \Leftrightarrow\ & \left[\boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[2]} - \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[1]}\right] \\ & \begin{bmatrix} \lambda^{[1]} \\ -\lambda^{[2]} \\ 1 \end{bmatrix} = \mathbf{0}. \end{aligned} \tag{4.60}$$

In other words, the homogeneous linear equation in (4.60), i.e.

$$\boldsymbol{M}\boldsymbol{z} \equiv \left[\boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[2]} - \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[1]}\right]\boldsymbol{z} = \mathbf{0}$$

has a non-trivial solution

$$\boldsymbol{z} = [\lambda^{[1]} \quad -\lambda^{[2]} \quad 1]^{\mathrm{T}}.$$

So the homogeneous matrix is rank-deficient and its determinant is zero, namely

$$\det \boldsymbol{M} = \left|\boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[2]} - \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[1]}\right| = 0. \tag{4.61}$$

Denote

$$\begin{aligned} \left[\boldsymbol{b}_1^{[1]} \quad \boldsymbol{b}_2^{[1]} \quad \boldsymbol{b}_3^{[1]}\right] &\equiv \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}, \text{i. e.} \quad \boldsymbol{b}_i^{[1]} \equiv \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}(\boldsymbol{A}^{[1]})^{-1}\boldsymbol{e}_i \quad (i \in \{1,2,3\}), \\ \left[\boldsymbol{b}_1^{[2]} \quad \boldsymbol{b}_2^{[2]} \quad \boldsymbol{b}_3^{[2]}\right] &\equiv \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}, \text{i. e.} \quad \boldsymbol{b}_j^{[2]} \equiv \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}(\boldsymbol{A}^{[2]})^{-1}\boldsymbol{e}_j \quad (j \in \{1,2,3\}), \\ \boldsymbol{b} &\equiv \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[2]} - \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}^{[1]} = \boldsymbol{T}_{\mathrm{cw}}^{[1]} - \boldsymbol{T}_{\mathrm{cw}}^{[2]} \end{aligned}$$

for (4.61). In the last equation above, $\boldsymbol{b}$ denotes the vector $\overrightarrow{\boldsymbol{O}_{\mathrm{c}}^{[2]}\boldsymbol{O}_{\mathrm{c}}^{[1]}}$, namely the vector connecting from the second camera coordinates system origin to the first camera coordinates system origin, expressed in the world coordinates system. Then

$$\begin{aligned}\det \boldsymbol{M} = & \left|u^{[1]}\boldsymbol{b}_1^{[1]} + v^{[1]}\boldsymbol{b}_2^{[1]} + \boldsymbol{b}_3^{[1]} \quad u^{[2]}\boldsymbol{b}_1^{[2]} + v^{[2]}\boldsymbol{b}_2^{[2]} + \boldsymbol{b}_3^{[2]} \quad \boldsymbol{b}\right| \\ = & \; u^{[1]}u^{[2]}\left|\boldsymbol{b}_1^{[1]} \quad \boldsymbol{b}_1^{[2]} \quad \boldsymbol{b}\right| + u^{[1]}v^{[2]}\left|\boldsymbol{b}_1^{[1]} \quad \boldsymbol{b}_2^{[2]} \quad \boldsymbol{b}\right| + u^{[1]}\left|\boldsymbol{b}_1^{[1]} \quad \boldsymbol{b}_3^{[2]} \quad \boldsymbol{b}\right| + \\ & v^{[1]}u^{[2]}\left|\boldsymbol{b}_2^{[1]} \quad \boldsymbol{b}_1^{[2]} \quad \boldsymbol{b}\right| + v^{[1]}v^{[2]}\left|\boldsymbol{b}_2^{[1]} \quad \boldsymbol{b}_2^{[2]} \quad \boldsymbol{b}\right| + v^{[1]}\left|\boldsymbol{b}_2^{[1]} \quad \boldsymbol{b}_3^{[2]} \quad \boldsymbol{b}\right| + \\ & u^{[2]}\left|\boldsymbol{b}_3^{[1]} \quad \boldsymbol{b}_1^{[2]} \quad \boldsymbol{b}\right| + v^{[2]}\left|\boldsymbol{b}_3^{[1]} \quad \boldsymbol{b}_2^{[2]} \quad \boldsymbol{b}\right| + \left|\boldsymbol{b}_3^{[1]} \quad \boldsymbol{b}_3^{[2]} \quad \boldsymbol{b}\right| \\ = & \begin{bmatrix} u^{[1]} & v^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix},\end{aligned}$$

where

$$F_{ij} = \left|\boldsymbol{b}_i^{[1]} \quad \boldsymbol{b}_j^{[2]} \quad \boldsymbol{b}\right|, \quad i, j \in \{1, 2, 3\}.$$

Therefore, Eq. (4.61) is transformed equivalently into

$$\begin{bmatrix} u^{[1]} & v^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} = 0 \tag{4.62}$$

or compactly as

$$\left(\boldsymbol{P}_{\mathrm{i}}^{[1]A}\right)^{\mathrm{T}} \boldsymbol{F} \boldsymbol{P}_{\mathrm{i}}^{[2]A} = 0$$

which is right the formalism of the epipolar constraint. The matrix

$$\boldsymbol{F} \equiv \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \tag{4.63}$$

is called the **fundamental matrix** .

Note from (4.57) that

$$\lambda^{[k]} \boldsymbol{P}_{\mathrm{i}}^{[k]A} \equiv \lambda \begin{bmatrix} u^{[k]} \\ v^{[k]} \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha^{[k]} & \gamma^{[k]} & u_0^{[k]} \\ 0 & \beta^{[k]} & v_0^{[k]} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}}^{[k]} \\ y_{\mathrm{c}}^{[k]} \\ z_{\mathrm{c}}^{[k]} \end{bmatrix} \equiv \boldsymbol{A}^{[k]} \boldsymbol{P}_{\mathrm{c}}^{[k]} \tag{4.64}$$

where $k \in \{1, 2\}$ and

$$\boldsymbol{P}_{\mathrm{c}}^{[k]} \equiv \begin{bmatrix} x_{\mathrm{c}}^{[\mathrm{k}]} \\ y_{\mathrm{c}}^{[\mathrm{k}]} \\ z_{\mathrm{c}}^{[\mathrm{k}]} \end{bmatrix} = \left[\boldsymbol{R}_{\mathrm{wc}}^{[k]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}\right] \boldsymbol{P}_{\mathrm{w}}^{\mathrm{A}}.$$

Substitute (4.64) into (4.62) and obtain

$$\begin{bmatrix} x_{\mathrm{c}}^{[1]} & y_{\mathrm{c}}^{[1]} & z_{\mathrm{c}}^{[1]} \end{bmatrix} \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}}^{[2]} \\ y_{\mathrm{c}}^{[2]} \\ z_{\mathrm{c}}^{[2]} \end{bmatrix} = 0 \tag{4.65}$$

or compactly as

$$\boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{E} \boldsymbol{P}_{\mathrm{c}}^{[2]} = 0$$

which is another formalism of the epipolar constraint. The matrix

$$\boldsymbol{E} = \boldsymbol{A}^{[1]\mathrm{T}} \boldsymbol{F} \boldsymbol{A}^{[2]} = \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.66}$$

is called the **essential matrix** .

Fundamental and essential matrices are intrinsic
For a pair of cameras that form a stereo vision system, the mutual geometric relationship between the two camera coordinates systems is fixed, regardless of their poses in the world coordinates system. So

$$\boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{b} = \boldsymbol{R}_{\mathrm{wc}}^{[1]} (\boldsymbol{T}_{\mathrm{cw}}^{[1]} - \boldsymbol{T}_{\mathrm{cw}}^{[2]})$$

and

$$\boldsymbol{R}_{\mathrm{wc}}^{[2]} \boldsymbol{b} = \boldsymbol{R}_{\mathrm{wc}}^{[2]} (\boldsymbol{T}_{\mathrm{cw}}^{[1]} - \boldsymbol{T}_{\mathrm{cw}}^{[2]}),$$

which mean the vector $\overrightarrow{\boldsymbol{O}_{\mathrm{c}}^{[2]} \boldsymbol{O}_{\mathrm{c}}^{[1]}}$ expressed in the first and second camera coordinates systems respectively, are both fixed. Besides, the concatenated rotation matrix

$$\boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}},$$

which means the rotation of the second camera coordinates system with respect to the first camera coordinates system, is also fixed.

Consider a generic element F_{ij} $(i, j \in \{1, 2, 3\})$ in the fundamental matrix

$$\begin{aligned} F_{ij} &= \left| \boldsymbol{b}_i^{[1]} \quad \boldsymbol{b}_j^{[2]} \quad \boldsymbol{b} \right| = \left| \boldsymbol{R}_{\mathrm{wc}}^{[1]} \right| \cdot \left| \boldsymbol{b}_i^{[1]} \quad \boldsymbol{b}_j^{[2]} \quad \boldsymbol{b} \right| \quad (\text{note} \left| \boldsymbol{R}_{\mathrm{wc}}^{[1]} \right| = 1) \\ &= \left| \boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}} (\boldsymbol{A}^{[1]})^{-1} \boldsymbol{e}_i \quad \boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{R}_{\mathrm{wc}}^{[2]\mathrm{T}} \left(\boldsymbol{A}^{[2]}\right)^{-1} \boldsymbol{e}_j \quad \boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{b} \right| \\ &= \left| (\boldsymbol{A}^{[1]})^{-1} \boldsymbol{e}_i \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} \left(\boldsymbol{A}^{[2]}\right)^{-1} \boldsymbol{e}_j \quad \boldsymbol{R}_{\mathrm{wc}}^{[1]} \boldsymbol{b} \right| \end{aligned} \tag{4.67}$$

is constant because all $\left(\boldsymbol{A}^{[1]}\right)^{-1}$, $\boldsymbol{R}_{\mathrm{cc}}{}^{[2,1]}$, $\left(\boldsymbol{A}^{[2]}\right)^{-1}$, and $\boldsymbol{R}_{\mathrm{wc}}{}^{[1]} \boldsymbol{b}$ are fixed. Since F_{ij} is generic in above analysis, the fundamental matrix is constant once the internal configuration of the stereo vision system is settled down, namely once the relative pose between the two cameras of the stereo vision system and their respective intrinsic parameters are determined. Therefore, for the stereo vision system, the fundamental matrix is a set of intrinsic parameters that reflect its inherent property, regardless of its external configuration which is characterized by its extrinsic parameters with respect to the world coordinates system. By (4.66) one can know that the essential matrix is also intrinsic.

It is worth noting that the epipolar constraint (4.62) still holds if the fundamental matrix is scaled by an arbitrary scalar factor. In other words, a fundamental matrix, even scaled by an arbitrary scalar factor, still characterizes the same epipolar relationship between the pair of cameras that form stereo vision. Bearing this in mind, one had better not treat the invariance of the fundamental matrix rigidly, but treat such invariance somehow like how one treats perspective mapping, homography, and some other relationships in projective geometry and how one treats the invariance of the perspective mapping matrix, the homography matrix, etc. that characterize these relationships.

Fundamental and essential matrices are rank-deficient
Let

$$\boldsymbol{a} \equiv \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1^{[2]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b}_3^{[2]} \end{bmatrix}^{-1} \boldsymbol{b} = \boldsymbol{A}^{[2]} \boldsymbol{R}_{\mathrm{wc}}{}^{[2]} \boldsymbol{b}$$

which is a non-zero vector. So

$$\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j = \begin{bmatrix} \boldsymbol{b}_1^{[2]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b}_3^{[2]} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \boldsymbol{b}$$

and

$$\begin{aligned} \boldsymbol{F}\boldsymbol{a} &= \begin{bmatrix} \det\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b}_1^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b}_3^{[2]} & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_2^{[1]} & \boldsymbol{b}_1^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_2^{[1]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_2^{[1]} & \boldsymbol{b}_3^{[2]} & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_3^{[1]} & \boldsymbol{b}_1^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_3^{[1]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b} \end{bmatrix} & \det\begin{bmatrix} \boldsymbol{b}_3^{[1]} & \boldsymbol{b}_3^{[2]} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \\ &= \begin{bmatrix} \det\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_2^{[1]} & \sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_3^{[1]} & \sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \det\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_2^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \\ \det\begin{bmatrix} \boldsymbol{b}_3^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \boldsymbol{0}. \end{aligned} \tag{4.68}$$

Treat (4.68) as an equation that has a non-trivial solution $\boldsymbol{a}$, which implies that the fundamental matrix $\boldsymbol{F}$ is rank-deficient.

Note again the form of (4.61) for which $\begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b}_2^{[1]} & \boldsymbol{b}_3^{[1]} \end{bmatrix}$ and $\begin{bmatrix} \boldsymbol{b}_1^{[2]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b}_3^{[2]} \end{bmatrix}$ are defined, one has

$$\left| \boldsymbol{b}_1^{[1]} \quad \boldsymbol{b}_2^{[1]} \quad \boldsymbol{b}_3^{[1]} \right| = \left| \boldsymbol{R}_{\mathrm{wc}}^{[1]\mathrm{T}} (\boldsymbol{A}^{[1]})^{-1} \right| = 1 / \left| \boldsymbol{A}^{[1]} \right| \neq 0,$$

which implies that $\boldsymbol{b}_1^{[1]}$, $\boldsymbol{b}_2^{[1]}$, and $\boldsymbol{b}_3^{[1]}$ are not coplanar. Besides, one has

$$\left|\boldsymbol{b}_1^{[2]} \quad \boldsymbol{b}_2^{[2]} \quad \boldsymbol{b}_3^{[2]}\right| = \left|\boldsymbol{R}_{\mathrm{wc}}^{[2]}\mathrm{T}(\boldsymbol{A}^{[2]})^{-1}\right| = 1/\left|\boldsymbol{A}^{[2]}\right| \neq 0,$$

which implies that $\boldsymbol{b}_1^{[2]}$, $\boldsymbol{b}_2^{[2]}$, and $\boldsymbol{b}_3^{[2]}$ are not coplanar either.

If $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j \neq \lambda \boldsymbol{b}$ (λ is a scaling factor), i.e. if $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j$ and $\boldsymbol{b}$ are not colinear, then there must be certain $\boldsymbol{b}_i^{[1]}$ ($i \in \{1, 2, 3\}$) such that $\boldsymbol{b}_i^{[1]}$, $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j$, and $\boldsymbol{b}$ are not coplanar. Otherwise, $\boldsymbol{b}_1^{[1]}$, $\boldsymbol{b}_2^{[1]}$, and $\boldsymbol{b}_3^{[1]}$ must be all located on the plane spanned by $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j$ and $\boldsymbol{b}$, which contradicts the fact that $\boldsymbol{b}_1^{[1]}$, $\boldsymbol{b}_2^{[1]}$, and $\boldsymbol{b}_3^{[1]}$ are not coplanar.

The fact that $\boldsymbol{b}_i^{[1]}$, $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j$, and $\boldsymbol{b}$ are not coplanar further implies that

$$\det\begin{bmatrix}\boldsymbol{b}_i^{[1]} & \displaystyle\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b}\end{bmatrix} \neq 0 \Rightarrow \boldsymbol{F}\boldsymbol{a} = \begin{bmatrix} \det\begin{bmatrix}\boldsymbol{b}_1^{[1]} & \displaystyle\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b}\end{bmatrix} \\ \det\begin{bmatrix}\boldsymbol{b}_2^{[1]} & \displaystyle\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b}\end{bmatrix} \\ \det\begin{bmatrix}\boldsymbol{b}_3^{[1]} & \displaystyle\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j & \boldsymbol{b}\end{bmatrix} \end{bmatrix} \neq \boldsymbol{0}.$$

In other words, $\sum_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j = \lambda \boldsymbol{b}$ is the only form of solution of the equation

$$\boldsymbol{F}\boldsymbol{a} = \boldsymbol{0}$$

and hence the kernel space of the fundamental matrix $\boldsymbol{F}$ is one-dimensional. Therefore, the rank of the fundamental matrix $\boldsymbol{F}$ is right two. By (4.66) one can know that the essential matrix is also rank-deficient and is right two.

Derivation of the essential matrix

Consider the epipolar constraint (4.65)

$$\begin{bmatrix} x_{\mathrm{c}}^{[1]} & y_{\mathrm{c}}^{[1]} & z_{\mathrm{c}}^{[1]} \end{bmatrix} \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}}^{[2]} \\ y_{\mathrm{c}}^{[2]} \\ z_{\mathrm{c}}^{[2]} \end{bmatrix} = 0,$$

i.e.

$$\boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{E} \boldsymbol{P}_{\mathrm{c}}^{[2]} = 0.$$

Let $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$ and $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ denote respectively the rotation matrix and translation vector of the second camera coordinates system with respect to the first camera coordinates system. Let $\boldsymbol{R}_{\mathrm{cc}}^{[1,2]}$ and $\boldsymbol{T}_{\mathrm{cc}}^{[1,2]}$ denote respectively the rotation matrix and translation vector of the first camera coordinates system with respect to the second camera coordinates system. The following relationships hold

$$\boldsymbol{R}_{\mathrm{cc}}^{[1,2]} = \boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}, \quad \boldsymbol{T}_{\mathrm{cc}}^{[1,2]} = -\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}.$$

Substitute

$$\boldsymbol{P}_{\mathrm{c}}^{[1]} = \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$$

into (4.65) and obtain

$$\left(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]}\right)^{\mathrm{T}}\boldsymbol{E}\boldsymbol{P}_{\mathrm{c}}^{[2]} = \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\boldsymbol{P}_{\mathrm{c}}^{[2]} = 0 \tag{4.69}$$

which holds for arbitrary $\boldsymbol{P}_{\mathrm{c}}^{[2]}$ and hence for arbitrary $\boldsymbol{P}_{\mathrm{c}}^{[2]} + \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}$ namely $\boldsymbol{P}_{\mathrm{c}}^{[2]}$ with infinitesimal variation $\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}$. Then substitute $\boldsymbol{P}_{\mathrm{c}}^{[2]} + \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}$ into (4.69) and obtain

$$\begin{aligned}
& \left(\boldsymbol{P}_{\mathrm{c}}^{[2]} + \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}\right)^{\mathrm{T}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}(\boldsymbol{P}_{\mathrm{c}}^{[2]} + \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}) + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}(\boldsymbol{P}_{\mathrm{c}}^{[2]} + \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]}) = 0 \\
\Leftrightarrow\ & \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} = 0 \\
\Leftrightarrow\ & \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E})^{\mathrm{T}}\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} = 0 \\
\Leftrightarrow\ & [\boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E})^{\mathrm{T}} + \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}]\Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} = 0 \\
& (\text{note that } \Delta\boldsymbol{P}_{\mathrm{c}}^{[2]} \text{ can be arbitrary}) \\
\Leftrightarrow\ & \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E})^{\mathrm{T}} + \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} \equiv 0 \\
\Leftrightarrow\ & \boldsymbol{P}_{\mathrm{c}}^{[2]\mathrm{T}}[(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E})^{\mathrm{T}} + \boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}] + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} = 0 \\
& (\text{note that } \boldsymbol{P}_{\mathrm{c}}^{[2]} \text{ can be arbitrary})
\end{aligned}$$

which finally leads to the following equalities in terms of the essential matrix $\boldsymbol{E}$

$$(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E})^{\mathrm{T}} + \boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} \equiv 0, \tag{4.70a}$$

$$\boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} \equiv 0. \tag{4.70b}$$

The first equation of (4.70) implies that $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E}$ is a skew-symmetric matrix. Suppose

$$\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{E} = \boldsymbol{S}_{\boldsymbol{w}} \equiv \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} \Longleftrightarrow \boldsymbol{E} = \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{S}_{\boldsymbol{w}}, \tag{4.71}$$

where $\boldsymbol{S}_{\boldsymbol{w}}$ denotes the skew-symmetric matrix[16] of certain vector $\boldsymbol{w} \equiv [w_x \quad w_y \quad w_z]^{\mathrm{T}}$. Substitute (4.71) into the second equation of (4.70) and obtain

$$\boldsymbol{T}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{S}_{\boldsymbol{w}} = 0 \Longleftrightarrow \boldsymbol{S}_{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} = 0 \Longleftrightarrow -\boldsymbol{w} \times (\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}) = 0$$

which implies that $\boldsymbol{w}$ and $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ are colinear, namely

$$\boldsymbol{w} \propto \boldsymbol{R}_{\mathrm{cc}}^{[2,1]\mathrm{T}}\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} = -\boldsymbol{T}_{\mathrm{cc}}^{[1,2]}. \tag{4.72}$$

Substitute (4.72) into (4.71) and obtain

$$\begin{aligned}\boldsymbol{E} &\propto \boldsymbol{R}_{\text{cc}}^{[2,1]}\boldsymbol{S}_{\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{T}_{\text{cc}}^{[2,1]}} = \boldsymbol{R}_{\text{cc}}^{[2,1]}\boldsymbol{S}_{\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{T}_{\text{cc}}^{[2,1]}}(\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{R}_{\text{cc}}^{[2,1]}) \\ &= \boldsymbol{R}_{\text{cc}}^{[2,1]}(\boldsymbol{S}_{\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{T}_{\text{cc}}^{[2,1]}}\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}})\boldsymbol{R}_{\text{cc}}^{[2,1]} \\ &= \boldsymbol{R}_{\text{cc}}^{[2,1]}(\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}})\boldsymbol{R}_{\text{cc}}^{[2,1]} = \boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}}\boldsymbol{R}_{\text{cc}}^{[2,1]}\end{aligned} \tag{4.73}$$

as well as

$$\boldsymbol{E} \propto \boldsymbol{R}_{\text{cc}}^{[2,1]}\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[1,2]}} = \boldsymbol{R}_{\text{cc}}^{[1,2]\text{T}}\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[1,2]}}. \tag{4.74}$$

It is worth noting that derivation of (4.73) resorts to the fact that the cross product of two identically rotated vectors is the same to the identically rotated cross product of the two vectors. More specifically, given a rotation matrix $\boldsymbol{R}$ and two generic vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, then

$$\boldsymbol{S}_{\boldsymbol{Ra}}\boldsymbol{Rb} = (\boldsymbol{Ra}) \times (\boldsymbol{Rb}) = \boldsymbol{R}(\boldsymbol{a} \times \boldsymbol{b}) = \boldsymbol{RS}_{\boldsymbol{a}}\boldsymbol{b} \Longleftrightarrow \boldsymbol{S}_{\boldsymbol{Ra}}\boldsymbol{R} = \boldsymbol{RS}_{\boldsymbol{a}}.$$

If (4.65) or (4.69) is treated as a matrix equation in terms of the essential matrix $\boldsymbol{E}$, then (4.73) and (4.74) equivalently give the solution of the matrix equation.

Denote $\boldsymbol{T}_{\text{cc}}^{[2,1]} \equiv \begin{bmatrix} x_{\text{cc}}^{[2,1]} & y_{\text{cc}}^{[2,1]} & z_{\text{cc}}^{[2,1]} \end{bmatrix}^{\text{T}}$. According to (4.73), one has

$$\begin{aligned}\boldsymbol{E}\boldsymbol{E}^{\text{T}} &= \lambda^2\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}}\boldsymbol{R}_{\text{cc}}^{[2,1]}\boldsymbol{R}_{\text{cc}}^{[2,1]\text{T}}\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}}^{\text{T}} = \lambda^2\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}}\boldsymbol{S}_{\boldsymbol{T}_{\text{cc}}^{[2,1]}}^{\text{T}} \\ &= -\lambda^2\begin{bmatrix} 0 & -z_{\text{cc}}^{[2,1]} & y_{\text{cc}}^{[2,1]} \\ z_{\text{cc}}^{[2,1]} & 0 & -x_{\text{cc}}^{[2,1]} \\ -y_{\text{cc}}^{[2,1]} & x_{\text{cc}}^{[2,1]} & 0 \end{bmatrix}^2 \\ &= \lambda^2\begin{bmatrix} \left(y_{\text{cc}}^{[2,1]}\right)^2 + \left(z_{\text{cc}}^{[2,1]}\right)^2 & -x_{\text{cc}}^{[2,1]}y_{\text{cc}}^{[2,1]} & -x_{\text{cc}}^{[2,1]}z_{\text{cc}}^{[2,1]} \\ -x_{\text{cc}}^{[2,1]}y_{\text{cc}}^{[2,1]} & \left(x_{\text{cc}}^{[2,1]}\right)^2 + \left(z_{\text{cc}}^{[2,1]}\right)^2 & -y_{\text{cc}}^{[2,1]}z_{\text{cc}}^{[2,1]} \\ -x_{\text{cc}}^{[2,1]}z_{\text{cc}}^{[2,1]} & -y_{\text{cc}}^{[2,1]}z_{\text{cc}}^{[2,1]} & \left(x_{\text{cc}}^{[2,1]}\right)^2 + \left(y_{\text{cc}}^{[2,1]}\right)^2 \end{bmatrix}\end{aligned}$$

and

$$\det\,(s\boldsymbol{I} - \boldsymbol{E}\boldsymbol{E}^{\text{T}}) = s\left[s - \lambda^2\left(x_{\text{cc}}^{[2,1]}\right)^2 - \lambda^2\left(y_{\text{cc}}^{[2,1]}\right)^2 - \lambda^2\left(z_{\text{cc}}^{[2,1]}\right)^2\right]^2 = s(s - \lambda^2\|\boldsymbol{T}_{\text{cc}}^{[2,1]}\|_2^2)^2.$$

So $\boldsymbol{E}\boldsymbol{E}^{\text{T}}$ has an eigenvalue 0 and two equivalent eigenvalues $\lambda^2\|\boldsymbol{T}_{\text{cc}}^{[2,1]}\|_2^2$. If one performs singular value decomposition of $\boldsymbol{E}$, then the decomposition must be of the following form $\boldsymbol{E} = \boldsymbol{U}\text{diag}(\sigma, \sigma, 0)\boldsymbol{V}^{\text{T}}$ or in details as

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 & \boldsymbol{U}_3 \end{bmatrix}\begin{bmatrix} \sigma & & \\ & \sigma & \\ & & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{V}_1^{\text{T}} \\ \boldsymbol{V}_2^{\text{T}} \\ \boldsymbol{V}_3^{\text{T}} \end{bmatrix} \tag{4.75}$$

where $\sigma = \sqrt{\lambda^2 \|\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}\|_2^2} = \lambda \|\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}\|_2$.

Epipolar points
The projective point

$$\lambda_{a^{[2]}} \boldsymbol{a}^{[2]} \equiv \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1^{[2]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b}_3^{[2]} \end{bmatrix}^{-1} \boldsymbol{b} = \boldsymbol{A}^{[2]} \boldsymbol{R}_{\mathrm{wc}}^{[2]} \boldsymbol{b}$$

that satisfies

$$\boldsymbol{F}(\lambda_{a^{[2]}} \boldsymbol{a}^{[2]}) = \begin{bmatrix} \det \begin{bmatrix} \boldsymbol{b}_1^{[1]} & \sum\limits_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j^{[2]} & \boldsymbol{b} \end{bmatrix} \\ {[}12pt{]} \det \begin{bmatrix} \boldsymbol{b}_2^{[1]} & \sum\limits_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j^{[2]} & \boldsymbol{b} \end{bmatrix} \\ {[}12pt{]} \det \begin{bmatrix} \boldsymbol{b}_3^{[1]} & \sum\limits_{j=1}^{3} \boldsymbol{b}_j^{[2]} a_j^{[2]} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \det \begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \\ \det \begin{bmatrix} \boldsymbol{b}_2^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \\ \det \begin{bmatrix} \boldsymbol{b}_3^{[1]} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \mathbf{0}$$

and the projective point

$$\lambda_{a^{[1]}} \boldsymbol{a}^{[1]} \equiv \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1^{[1]} & \boldsymbol{b}_2^{[1]} & \boldsymbol{b}_3^{[1]} \end{bmatrix}^{-1} \boldsymbol{b} = \boldsymbol{A}^{[1]} \boldsymbol{R}_{\mathrm{wc}^{[1]}} \boldsymbol{b}$$

that satisfies

$$\boldsymbol{F}^{\mathrm{T}}(\lambda_{a^{[1]}} \boldsymbol{a}^{[1]}) = \begin{bmatrix} \det \begin{bmatrix} \sum\limits_{i=1}^{3} \boldsymbol{b}_i^{[1]} a_i^{[1]} & \boldsymbol{b}_1^{[2]} & \boldsymbol{b} \end{bmatrix} \\ {[}12pt{]} \det \begin{bmatrix} \sum\limits_{i=1}^{3} \boldsymbol{b}_i^{[1]} a_i^{[1]} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b} \end{bmatrix} \\ {[}12pt{]} \det \begin{bmatrix} \sum\limits_{i=1}^{3} \boldsymbol{b}_i^{[1]} a_i^{[1]} & \boldsymbol{b}_3^{[2]} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \det \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b}_1^{[2]} & \boldsymbol{b} \end{bmatrix} \\ \det \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b}_2^{[2]} & \boldsymbol{b} \end{bmatrix} \\ \det \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b}_3^{[2]} & \boldsymbol{b} \end{bmatrix} \end{bmatrix} = \mathbf{0}$$

are called **epipolar points** or **epipoles**.

Epipolar lines
According to the epipolar constraint (4.62), given a known point $(u^{[1]}, v^{[1]})$ in the image coordinates system of the first camera, its counterpart in the image coordinates system of the second camera must be located on the line

$$\begin{aligned} L^{[1]}: \quad (F_{11}u^{[1]} + F_{21}v^{[1]} + F_{31})u + & \; (F_{12}u^{[1]} + F_{22}v^{[1]} + F_{32})v + \\ & \; (F_{13}u^{[1]} + F_{23}v^{[1]} + F_{33}) = 0. \end{aligned} \tag{4.76}$$

On the other hand, given a known point $(u^{[2]}, v^{[2]})$ in the image coordinates system of the second camera, its counterpart in the image coordinates system of the first camera must be located on the line

$$\begin{aligned} L^{[2]}: \quad (F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13})u + & \; (F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23})v + \\ & \; (F_{31}u^{[2]} + F_{32}v^{[2]} + F_{33}) = 0. \end{aligned} \tag{4.77}$$

The line $L^{[1]}$ specified in (4.76) and the line $L^{[2]}$ specified in (4.77) are mutually **epipolar lines** of each other. To better understand such mutual relationship between epipolar lines, duality between (4.76) and (4.77) need to be further clarified.

Consider two different points $(u_1^{[1]}, v_1^{[1]})$ and $(u_2^{[1]}, v_2^{[1]})$ located on the same epipolar line $L^{[2]}$ in the image coordinates system of the first camera, satisfying

$$\begin{aligned} (F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13})u_1^{[1]} + & \; (F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23})v_1^{[1]} + \\ & \; (F_{31}u^{[2]} + F_{32}v^{[2]} + F_{33}) = 0, \\ (F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13})u_2^{[1]} + & \; (F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23})v_2^{[1]} + \\ & \; (F_{31}u^{[2]} + F_{32}v^{[2]} + F_{33}) = 0. \end{aligned}$$

Since $\mathrm{rank}(\boldsymbol{F}) = 2$, one can express (F_{31}, F_{32}, F_{33}) as a linear combination of (F_{11}, F_{12}, F_{13}) and (F_{21}, F_{22}, F_{23})

$$(F_{31}, F_{32}, F_{33}) = \mu_1(F_{11}, F_{12}, F_{13}) + \mu_2(F_{21}, F_{22}, F_{23}).$$

So

$$\begin{aligned} (F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13})(u_1^{[1]} + \mu_1) + (F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23})(v_1^{[1]} + \mu_2) &= 0, \\ (F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13})(u_2^{[1]} + \mu_1) + (F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23})(v_2^{[1]} + \mu_2) &= 0. \end{aligned}$$

and

$$\begin{aligned} u_1^{[1]} + \mu_1 &= \lambda_1(F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23}), \quad & v_1^{[1]} + \mu_2 &= -\lambda_1(F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13}), \\ u_2^{[1]} + \mu_1 &= \lambda_2(F_{21}u^{[2]} + F_{22}v^{[2]} + F_{23}), \quad & v_2^{[1]} + \mu_2 &= -\lambda_2(F_{11}u^{[2]} + F_{12}v^{[2]} + F_{13}). \end{aligned}$$

The epipolar lines of $(u_1^{[1]}, v_1^{[1]})$ and $(u_2^{[1]}, v_2^{[1]})$ in the image coordinates system of the second camera are respectively

$$\begin{aligned} L_1^{[1]}: \quad (F_{11}u_1^{[1]} + F_{21}v_1^{[1]} + F_{31})u + & \; (F_{12}u_1^{[1]} + F_{22}v_1^{[1]} + F_{32})v + \\ & \; (F_{13}u_1^{[1]} + F_{23}v_1^{[1]} + F_{33}) = 0, \\ L_2^{[1]}: \quad (F_{11}u_2^{[1]} + F_{21}v_2^{[1]} + F_{31})u + & \; (F_{12}u_2^{[1]} + F_{22}v_2^{[1]} + F_{32})v + \\ & \; (F_{13}u_2^{[1]} + F_{23}v_2^{[1]} + F_{33}) = 0. \end{aligned}$$

Note that

$$\begin{aligned}
& F_{11}u_1^{[1]}+F_{21}v_1^{[1]}+F_{31} \\
= \; & F_{11}(u_1^{[1]}+\mu_1)+F_{21}(v_1^{[1]}+\mu_2) \quad (\mathrm{as}\, \mathrm{F}_{31}=\mathrm{F}_{11}\mu_1+\mathrm{F}_{21}\mu_2) \\
= \; & F_{11}\cdot\lambda_1(F_{21}u^{[2]}+F_{22}v^{[2]}+F_{23})-F_{21}\cdot\lambda_1(F_{11}u^{[2]}+F_{12}v^{[2]}+F_{13}) \\
= \; & \lambda_1[(F_{11}F_{22}-F_{21}F_{12})v^{[2]}+F_{11}F_{23}-F_{21}F_{13}], \\
& F_{11}u_2^{[1]}+F_{21}v_2^{[1]}+F_{31} \\
= \; & F_{11}(u_2^{[1]}+\mu_1)+F_{21}(v_2^{[1]}+\mu_2) \\
= \; & F_{11}\cdot\lambda_2(F_{21}u^{[2]}+F_{22}v^{[2]}+F_{23})-F_{21}\cdot\lambda_2(F_{11}u^{[2]}+F_{12}v^{[2]}+F_{13}) \\
= \; & \lambda_2[(F_{11}F_{22}-F_{21}F_{12})v^{[2]}+F_{11}F_{23}-F_{21}F_{13}]
\end{aligned}$$

and hence

$$F_{11}u_1^{[1]}+F_{21}v_1^{[1]}+F_{31}:F_{11}u_2^{[1]}+F_{21}v_2^{[1]}+F_{31}=\lambda_1:\lambda_2.$$

Similarly one has

$$\begin{aligned}
F_{12}u_1^{[1]}+F_{22}v_1^{[1]}+F_{32}:F_{12}u_2^{[1]}+F_{22}v_2^{[1]}+F_{32} &= \lambda_1:\lambda_2, \\
F_{13}u_1^{[1]}+F_{23}v_1^{[1]}+F_{33}:F_{13}u_2^{[1]}+F_{23}v_2^{[1]}+F_{33} &= \lambda_1:\lambda_2.
\end{aligned}$$

Therefore, the lines $L_1^{[1]}$ and $L_2^{[1]}$ are identical.

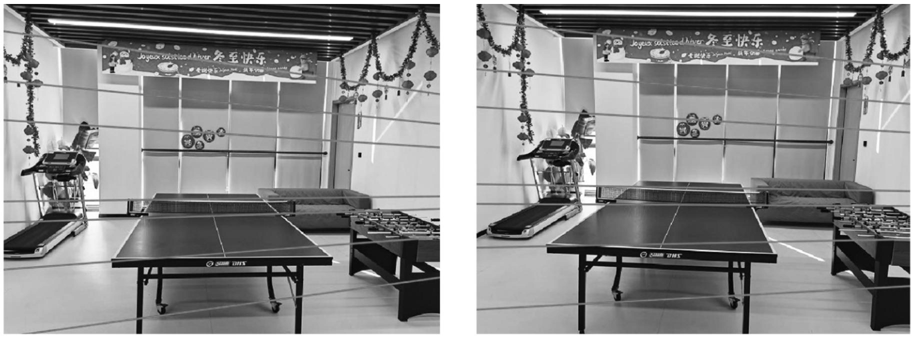
Above conclusion reflects the *duality of epipolar lines* (4.76) and (4.77): For any point on $L^{[2]}$ in the image coordinates system of the first camera, its epipolar line in the image coordinates system of the second camera is identical to the epipolar line $L^{[1]}$. On the other hand, similarly for any point on $L^{[1]}$ in the image coordinates system of the second camera, its epipolar line in the image coordinates system of the first camera is identical to the epipolar line $L^{[2]}$.

An example of stereo vision images with several corresponding pairs of epipolar lines on them respectively is illustrated in Fig. 4.17. The illustrated epipolar lines intuitively interpret the sense of the epipolar constraint:

any point in the environment cannot appear freely in both images.

When its location in one image is on a specific epipolar line, its location in the other image must be constrained onto the corresponding

epipolar line, without any chance to be located elsewhere.



(a) (b)

Fig. 4.17 Epipolar constraint **a** epipolar lines in the left image; **b** epipolar lines in the right image

4.5.3 Rectified Stereo Vision

An indispensable but difficult part of stereo vision processing is to match the pair of images that form stereo vision, namely to determine correspondences of pixels between the pair of images. Given a generic pixel in one image, to search for its corresponding pixel in the other image, one does not need at all to consider all pixels in the other image. Thanks to the epipolar constraint, one can fairly confine the search only to the corresponding epipolar line, which largely reduces the amount of potential pixels to be considered.

To further facilitate handling of such stereo vision matching problem, one may rectify the pair of images such that epipolar lines in the rectified images are geometrically regular (usually horizontally aligned), instead of inclined irregularly in the original images. The conventional way of rectifying stereo vision is to transform the original pair of images moderately via homography [24, 25].

Denote the rectifying homography matrices associated with the first and second images as $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ respectively. The basic spirit of the conventional way is to configure $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ such that

$$\boldsymbol{H}^{[1]\mathrm{T}}\boldsymbol{F}\boldsymbol{H}^{[2]} = \boldsymbol{F}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -s \\ 0 & s & 0 \end{bmatrix} \tag{4.78}$$

where s is certain non-zero scaling factor. Denote the rectified image coordinates with the subscript "r" and formalize the relationship between the rectified image coordinates and the original image coordinates as

$$\begin{bmatrix} u^{[1]} \\ v^{[1]} \\ 1 \end{bmatrix} \propto \boldsymbol{H}^{[1]} \begin{bmatrix} u_\mathrm{r}^{[1]} \\ v_\mathrm{r}^{[1]} \\ 1 \end{bmatrix}, \quad \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} \propto \boldsymbol{H}^{[2]} \begin{bmatrix} u_\mathrm{r}^{[2]} \\ v_\mathrm{r}^{[2]} \\ 1 \end{bmatrix}. \tag{4.79}$$

Associate (4.62), (4.78), and (4.79) and obtain

$$\begin{aligned} s(v_\mathrm{r}^{[2]} - v_\mathrm{r}^{[1]}) &= \begin{bmatrix} u_\mathrm{r}^{[1]} & v_\mathrm{r}^{[1]} & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -s \\ 0 & s & 0 \end{bmatrix} \begin{bmatrix} u_\mathrm{r}^{[2]} \\ v_\mathrm{r}^{[2]} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} u_\mathrm{r}^{[1]} & v_\mathrm{r}^{[1]} & 1 \end{bmatrix} \boldsymbol{H}^{[1]\mathrm{T}}\boldsymbol{F}\boldsymbol{H}^{[2]} \begin{bmatrix} u_\mathrm{r}^{[2]} \\ v_\mathrm{r}^{[2]} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} u^{[1]} & v^{[1]} & 1 \end{bmatrix} \boldsymbol{F} \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} = 0 \Longleftrightarrow v_\mathrm{r}^{[2]} = v_\mathrm{r}^{[1]}. \end{aligned} \tag{4.80}$$

The equality derived in (4.80) conveys an important characteristic of the rectified epipolar constraint, namely each pair of corresponding epipolar lines are aligned onto the same horizontal line. This characteristic is especially desirable in practice because it largely facilitates handling of the stereo vision matching problem: to search for corresponding pixels, one only needs to scan the same rows in both image matrices.

The remaining question is how to configure $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ properly such that (4.78) is satisfied. Instead of handling (4.78) directly, one may first aim at configuring $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ such that the epipolar points in both images are rectified to horizontal infinity. This is equivalent to configuring $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ such that

$$\boldsymbol{F}^{\mathrm{T}}\boldsymbol{H}^{[1]}\begin{bmatrix}1\\0\\0\end{bmatrix} = \boldsymbol{0} \Longleftrightarrow \boldsymbol{F}^{\mathrm{T}}\boldsymbol{H}^{[1]}_{:,1} = \boldsymbol{0}, \tag{4.81a}$$

$$\boldsymbol{F}\boldsymbol{H}^{[2]}\begin{bmatrix}1\\0\\0\end{bmatrix} = \boldsymbol{0} \Longleftrightarrow \boldsymbol{F}\boldsymbol{H}^{[2]}_{:,1} = \boldsymbol{0}. \tag{4.81b}$$

For (4.81), note that the projective point $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^{\mathrm{T}}$ is right the projective point corresponding to horizontal infinity.

Perform *singular value decomposition* of $\boldsymbol{F}$ as $\boldsymbol{F} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{V}^{\mathrm{T}}$ or in details as

$$\boldsymbol{F} = \begin{bmatrix}\boldsymbol{U}_1 & \boldsymbol{U}_2 & \boldsymbol{U}_3\end{bmatrix}\begin{bmatrix}\sigma_1 & & \\ & \sigma_2 & \\ & & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_1^{\mathrm{T}}\\ \boldsymbol{V}_2^{\mathrm{T}}\\ \boldsymbol{V}_3^{\mathrm{T}}\end{bmatrix}$$

where $\sigma_1 \geqslant \sigma_2 > 0$. Then the first equation of (4.81) leads to

$$\begin{aligned}\boldsymbol{V}\mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{U}^{\mathrm{T}}\boldsymbol{H}^{[1]}_{:,1} = \boldsymbol{0} \quad &\Longleftrightarrow \mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{U}^{\mathrm{T}}\boldsymbol{H}^{[1]}_{:,1} = \boldsymbol{0}\\ &\Longleftrightarrow \begin{bmatrix}\sigma_1\boldsymbol{U}_1^{\mathrm{T}}\\ \sigma_2\boldsymbol{U}_2^{\mathrm{T}}\end{bmatrix}\boldsymbol{H}^{[1]}_{:,1} = \boldsymbol{0}\end{aligned}$$

which implies that $\boldsymbol{H}^{[1]}_{:,1}$ is orthogonal to $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ and hence

$$\boldsymbol{H}^{[1]}_{:,1} \propto \boldsymbol{U}_3. \tag{4.82}$$

Similarly, the second equation of (4.81) leads to

$$\begin{aligned}\boldsymbol{U}\mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{H}^{[2]}_{:,1} = \boldsymbol{0} \quad &\Longleftrightarrow \mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{V}^{\mathrm{T}}\boldsymbol{H}^{[2]}_{:,1} = \boldsymbol{0}\\ &\Longleftrightarrow \begin{bmatrix}\sigma_1\boldsymbol{V}_1^{\mathrm{T}}\\ \sigma_2\boldsymbol{V}_2^{\mathrm{T}}\end{bmatrix}\boldsymbol{H}^{[2]}_{:,1} = \boldsymbol{0}\end{aligned}$$

which implies that $\boldsymbol{H}^{[2]}_{:,1}$ is orthogonal to $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ and hence

$$\boldsymbol{H}^{[2]}_{:,1} \propto \boldsymbol{V}_3. \tag{4.83}$$

To satisfy (4.82) and (4.83), one may simply set $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ as

$$\boldsymbol{H}^{[1]} = \begin{bmatrix} \text{sign}(U_{13}) \cdot U_{13} & 0 & 0 \\ \text{sign}(U_{13}) \cdot U_{23} & 1 & 0 \\ \text{sign}(U_{13}) \cdot U_{33} & 0 & 1 \end{bmatrix}, \tag{4.84a}$$

$$\boldsymbol{H}^{[2]} = \begin{bmatrix} \text{sign}(V_{13}) \cdot V_{13} & 0 & 0 \\ \text{sign}(V_{13}) \cdot V_{23} & 1 & 0 \\ \text{sign}(V_{13}) \cdot V_{33} & 0 & 1 \end{bmatrix}. \tag{4.84b}$$

Use $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ described in (4.84) to transform the original fundamental matrix $\boldsymbol{F}$ into a new fundamental matrix

$$\boldsymbol{F}' = \boldsymbol{H}^{[1]\mathrm{T}} \boldsymbol{F} \boldsymbol{H}^{[2]}.$$

$$[1 \quad 0 \quad 0]\boldsymbol{F}' = \mathbf{0}, \quad \boldsymbol{F}' \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{0}$$

and hence must be of the form

$$\boldsymbol{F}' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{32} & m_{33} \end{bmatrix} \equiv \begin{bmatrix} 0 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \boldsymbol{M} \end{bmatrix}. \tag{4.85}$$

By so far, with the homography matrices $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ described in (4.84), the epipolar points in both images are rectified to horizontal infinity. In other words, epipolar lines in both images are rectified to horizontal lines. The next step is to rectify corresponding epipolar lines in both images to the same horizontal lines. For the purpose, one may simply fix the homography matrix $\boldsymbol{H}^{[1]}$ and multiply the homography matrix $\boldsymbol{H}^{[2]}$ by another homography matrix $\boldsymbol{H}_{\mathrm{c}}$ such that

$$\boldsymbol{F}'\boldsymbol{H}_{\mathrm{c}} = \boldsymbol{F}_0, \quad \text{i.e.} \quad \begin{bmatrix} 0 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \boldsymbol{M} \end{bmatrix} \boldsymbol{H}_{\mathrm{c}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -s \\ 0 & s & 0 \end{bmatrix}.$$

According to the block structure of above matrix equation, one may set $\boldsymbol{H}_{\mathrm{c}}$ in a form as

$$\boldsymbol{H}_{\mathrm{c}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & h_1 & h_3 \\ 0 & h_2 & 1 \end{bmatrix}.$$

Then

$$\begin{aligned} &\boldsymbol{M}\begin{bmatrix}h_1 & h_3\\ h_2 & 1\end{bmatrix} = \begin{bmatrix}0 & -s\\ s & 0\end{bmatrix} \Longleftrightarrow \mathrm{vec}\left(\boldsymbol{M}\begin{bmatrix}h_1 & h_3\\ h_2 & 1\end{bmatrix}\right) = \mathrm{vec}\left(\begin{bmatrix}0 & -s\\ s & 0\end{bmatrix}\right)\\ \Leftrightarrow \quad &(\boldsymbol{I}\otimes\boldsymbol{M})\begin{bmatrix}h_1\\ h_2\\ h_3\\ 1\end{bmatrix} = \begin{bmatrix}0\\ 1\\ -1\\ 0\end{bmatrix}s \Longleftrightarrow \begin{bmatrix}(\boldsymbol{I}\otimes\boldsymbol{M})_{:,1:3} & \begin{matrix}0\\ -1\\ 1\\ 0\end{matrix}\end{bmatrix}\begin{bmatrix}h_1\\ h_2\\ h_3\\ s\end{bmatrix} = -(\boldsymbol{I}\otimes\boldsymbol{M})_{:,4} \end{aligned} \tag{4.86}$$

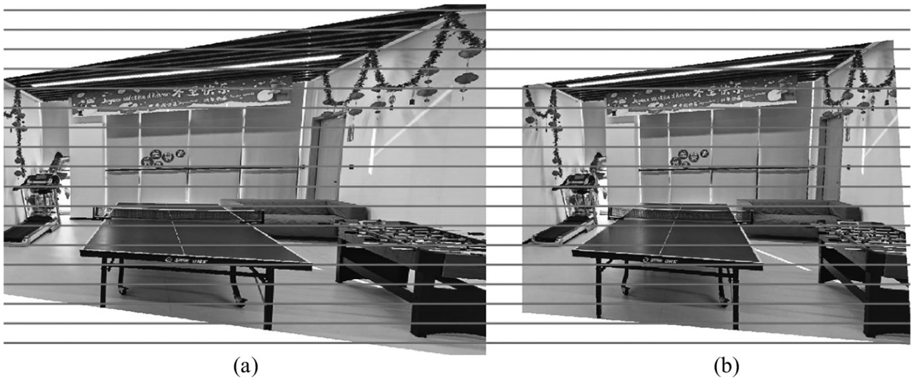
from which one can solve h_1, h_2, h_3 and obtain $\boldsymbol{H}_{\mathrm{c}}$. Finally, update $\boldsymbol{H}^{[2]}$ as

$$\boldsymbol{H}^{[2]} := \boldsymbol{H}^{[2]}\boldsymbol{H}_{\mathrm{c}}, \tag{4.87}$$

where := denotes *value assignment* rather than equality. Matlab code for above method of configuring the rectifying homography matrices $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ is given as follows.

RectifyFundamentalMatrix.m

```
%% Rectify the fundamental matrix between a pair of cameras
% Fm: fundamental matrix [uai;vai;1]’*Fm*[ubi;vbi;1] = 0
% Ha,Hb: Ha’*Fm*Hb = [0,0,0; 0,0,-s; 0,s,0]
function [Ha,Hb] = RectifyFundamentalMatrix(Fm)
    [U,S,V] = svd(Fm);
    Ha = eye(3); Ha(:,1) = sign(U(1,3))*U(:,3);
    Hb = eye(3); Hb(:,1) = sign(V(1,3))*V(:,3);
    Fm2 = Ha’*Fm*Hb; M = kron(eye(2),Fm2(2:3,2:3));
    A = [M(:,1:3),[0;-1;1;0]]; B = -M(:,4);
    hs = A\B; Hc = [1,0,0;0,hs(1),hs(3);0,hs(2),1];
    Hb = Hb*Hc;
end
```


(a)

(b)

Fig. 4.18 Rectified epipolar constraint **a** horizontal epipolar lines in the left image; **b** horizontal epipolar lines in the right image

Still take the pair of images illustrated in Fig. 4.17 as example. The effect of epipolar constraint rectification is demonstrated in Fig. 4.18. Corresponding epipolar lines in the original left and right images are consistently rectified to the same horizontal epipolar lines in the rectified images.

It is worth noting that configuration of the rectifying homography matrices $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ is not unique. There are infinite choices of $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ that satisfy (4.78). Besides, the conventional way of rectifying stereo vision via homography is based on the assumption that a moderate homography transform of the original image would still maintain the three-dimensional genuineness of objects in the image. The assumption can be fairly adopted in many practical applications, yet the homography transform does slightly incur deformation of the three-dimensional genuineness of objects in the image. Such deformation can be observed in Fig. 4.18. Therefore, besides above presented simple method of configuring the rectifying homography matrices $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$, a more sophisticated method is to configure $\boldsymbol{H}^{[1]}$ and $\boldsymbol{H}^{[2]}$ optimally, in terms of minimizing the deformation influence [25].

Commercial manufacturing
A conventional practice for manufacturing of a (binocular) stereo camera is to align the left and right cameras as well as possible at hardware level. So the extent of misalignment between the two cameras are unlikely to be such exaggerated as illustrated in Fig. 4.17, but tends to be slight, as illustrated in Fig. 4.19a, b. Readers may notice that the illustrated epipolar lines are slightly inclined and corresponding ones among them are not horizontally aligned.

To finally handle the slight misalignment remained after manufacturing at hardware level, one can resort to rectification at software level. The intuitive effect of stereo vision rectification is demonstrated in Fig. 4.19c, d. The slight misalignment is successfully rectified.

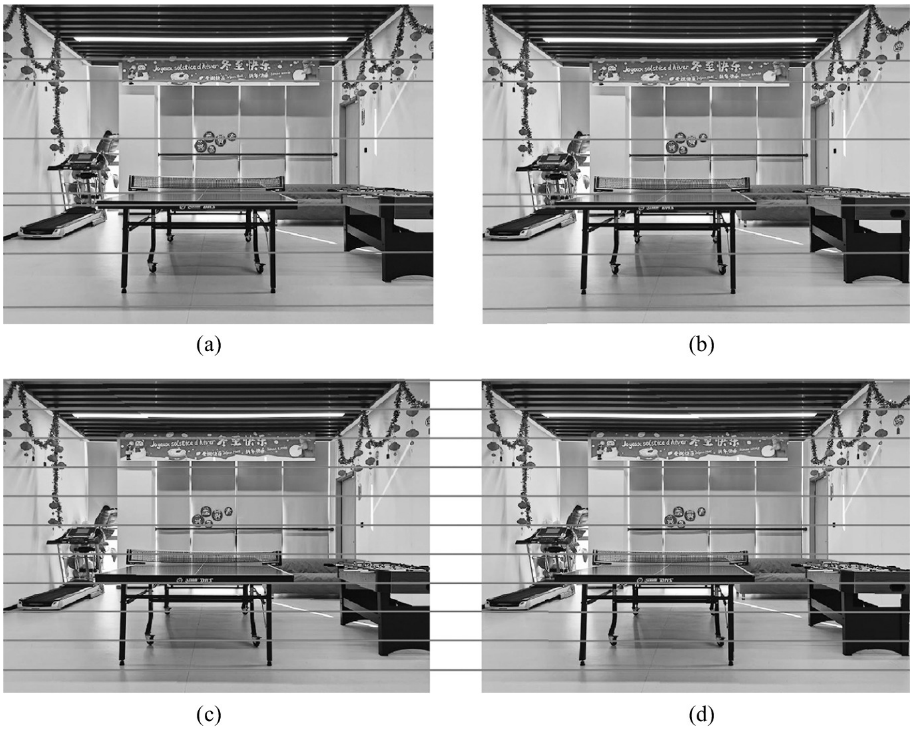


Fig. 4.19 Rectified epipolar constraint **a** raw left camera image; **b** raw right camera image; **c** rectified left camera image; **d** rectified right camera image

4.5.4 Binocular Disparity

The example pair of images illustrated in Fig. 4.17 is intentionally made to have a large view discrepancy. In practice, a stereo camera is usually manufactured in such a way that its left and right sub-vision systems are aligned in parallel configuration and the twin sub-vision systems differ from each other only by almost a horizontal translation. In other words, the stereo camera is manufactured to almost satisfy the *ideal stereo vision* condition that the twin sub-vision systems differ from each other only by a pure horizontal translation, as illustrated in Fig. 4.20. It is true that commercial manufacturing itself may not achieve the ideal stereo vision condition, yet with careful calibration and rectification, achievement of the ideal stereo vision condition can be fairly assumed.

For a stereo camera that is assumed to satisfy the ideal stereo vision condition illustrated in Fig. 4.20, the camera model can be much simplified. Take the left camera coordinates system as the common world coordinates system for both sub-vision systems. Then the camera model is formalized as

$$\begin{aligned} u^{[1]} &= \alpha \frac{x_{\mathrm{c}}^{[1]}}{z_{\mathrm{c}}^{[1]}} + u_0 = \alpha \frac{x_{\mathrm{w}}}{z_{\mathrm{w}}} + u0, \\ v^{[1]} &= \alpha \frac{y_{\mathrm{c}}^{[1]}}{z_{\mathrm{c}}^{[1]}} + v_0 = \alpha \frac{y_{\mathrm{w}}}{z_{\mathrm{w}}} + v0, \\ u^{[2]} &= \alpha \frac{x_{\mathrm{c}}^{[2]}}{z_{\mathrm{c}}^{[2]}} + u_0 = \alpha \frac{x_{\mathrm{w}}-b}{z_{\mathrm{w}}} + u0, \\ v^{[2]} &= \alpha \frac{y_{\mathrm{c}}^{[2]}}{z_{\mathrm{c}}^{[2]}} + v_0 = \alpha \frac{y_{\mathrm{w}}}{z_{\mathrm{w}}} + v0 = v^{[1]} \end{aligned}$$

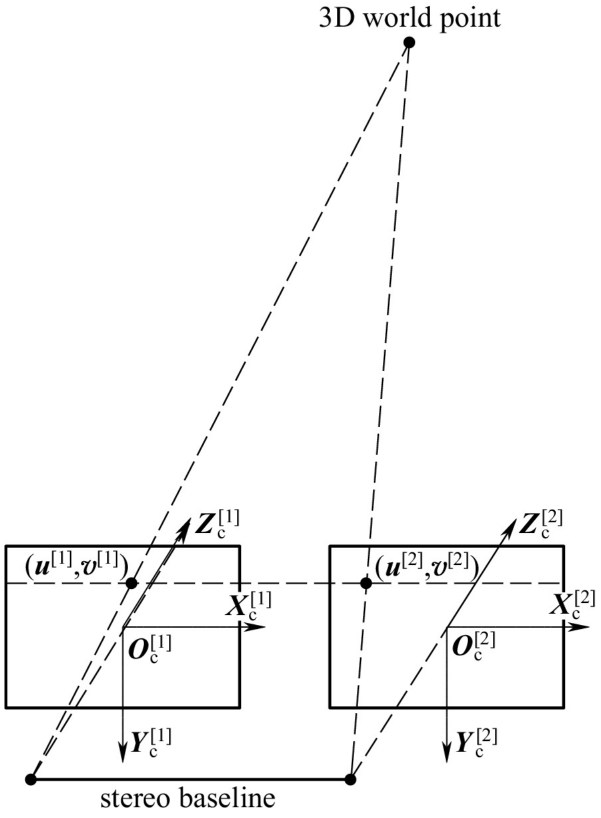

3D world point
$Z_{\mathrm{c}}^{[1]}$
$Z_{\mathrm{c}}^{[2]}$
$(u^{[1]},v^{[1]})$
$(u^{[2]},v^{[2]})$
$X_{\mathrm{c}}^{[1]}$
$X_{\mathrm{c}}^{[2]}$
$O_{\mathrm{c}}^{[1]}$
$O_{\mathrm{c}}^{[2]}$
$Y_{\mathrm{c}}^{[1]}$
$Y_{\mathrm{c}}^{[2]}$
stereo baseline

Fig. 4.20 Ideal stereo vision

namely

$$u^{[1]} = \alpha \frac{x_{\mathrm{w}}}{z_{\mathrm{w}}} + u0, \tag{4.88a}$$

$$u^{[2]} = \alpha \frac{x_{\mathrm{w}}-b}{z_{\mathrm{w}}} + u0, \tag{4.88b}$$

$$v^{[1]} = v^{[2]} \quad = \alpha \tfrac{y_{\mathrm{w}}}{z_{\mathrm{w}}} + v0, \tag{4.88c}$$

where b denotes the length of the *stereo baseline* and α denotes the camera intrinsic parameter α in the pinhole camera model .[17] The parameters α, u_0, v_0, and b are assumed to be known because they can be calibrated beforehand. From the camera model (4.88) one can derive the inverse camera model for ideal stereo vision as

$$x_{\mathrm{w}} \quad = \tfrac{(u^{[1]} - u_0) \cdot b}{u^{[1]} - u^{[2]}}, \tag{4.89a}$$

$$y_{\mathrm{w}} \quad = \tfrac{(v^{[1]} - v_0) \cdot b}{u^{[1]} - u^{[2]}}, \tag{4.89b}$$

$$z_{\mathrm{w}} \quad = \tfrac{\alpha \cdot b}{u^{[1]} - u^{[2]}}. \tag{4.89c}$$

The third equation of (4.89) conveys an important point of stereo vision: the longitudinal distance namely **depth** of a world point is inversely proportional to its horizontal coordinates difference in the pair of images. The horizontal coordinates difference $u^{[1]} - u^{[2]}$ is called the **binocular disparity** or simply **disparity** (also known as **parallax**) of the world point. For a generic world point, its depth depends only on its disparity, regardless of its visual locations in the images.

Based on this fact, one can generate a kind of image of the same size as the left image (or the right image), with each of its pixels storing the disparity of the same pixel in the left image. Such kind of image is called the **disparity image** or **disparity map** of the pair of stereo vision images. For example, a synthetic pair of stereo vision images is illustrated in Fig. 4.21a, b, where the scenario consists of a ground surface and a wall at certain distance in front.

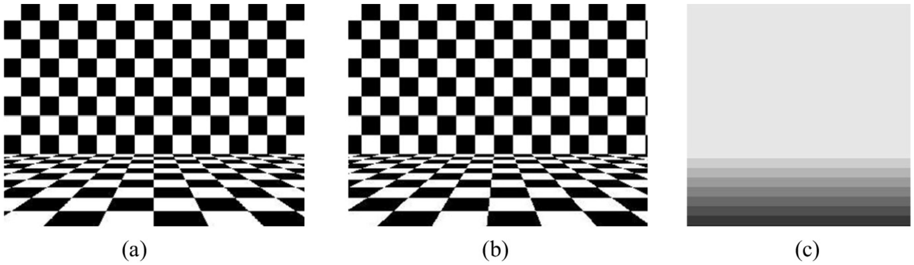

(a) (b) (c)

Fig. 4.21 Synthetic stereo vision **a** left stereo vision image; **b** right stereo vision image; **c** disparity image or disparity map

The disparity image is computed from the synthetic pair of stereo vision images and is visualized in Fig. 4.21c. For a point, the smaller its disparity is, i.e. the further it is, the brighter the corresponding pixel is. The disparity image provides a convenient way to encode and visualize depths of objects in the image. A representative method of disparity computation is the semiglobal matching method [26].

References

1. D. Hilbert. *The Foundations of Geometry* (Open Court Publishing Company, 1902)
2. W. Xiang. *Fundamental Geometry (in Chinese)* (People's Education Press, 2004)
3. O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint* (MIT Press, 1993)

4. H. Li, F. Nashashibi, *Lane detection (Part i): mono-vision based method*. INRIA Tech Report, RT-433 (2013)
5. H. Li, F. Nashashibi, Multi-vehicle cooperative perception and augmented reality for driver assistance: A possibility to see through front vehicle, in *IEEE International Conference on Intelligent Transportation Systems* (2011), pp. 242–247
6. H. Li, M. Tsukada, F. Nashashibi, M. Parent, Multivehicle cooperative local mapping: a methodology based on occupancy grid map merging. IEEE Trans. Intell. Transp. Syst. **15**(5), 2089–2100 (2014)
[Crossref]
7. G.H. Golub, C.F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, 1996)
8. M. Bertozzi, A. Broggi, A. Fascioli, Stereo inverse perspective mapping: theory and applications. Image Vis. Comput. **16**(8), 585–590 (1998)
[Crossref]
9. M. Bertozzi, A. Broggi, Gold: a parallel real-time stereo vision system for generic obstacle and lane detection. IEEE Trans. Image Process. **7**(1), 62–81 (1998)
[ADS][Crossref][PubMed]
10. H. Li, F. Nashashibi, Robust real-time lane detection based on lane mark segment features and general a priori knowledge, in *IEEE International Conference on Robotics and Biomimetics* (2011), pp. 812–817
11. T. Bruls, H. Porav, L. Kunze, P. Newman, The right (angled) perspective: improving the understanding of road scenes using boosted inverse perspective mapping, in *IEEE Intelligent Vehicles Symposium* (2019), , pp. 302–309
12. C. Prakash, F. Akhbari, L. Karam, Robust obstacle detection for advanced driver assistance systems using distortions of inverse perspective mapping of a monocular camera. Robot. Autonom. Syst. **114**, 172–186 (2019)
[Crossref]
13. Y. Kim, D. Kum, Deep learning based vehicle position and orientation estimation via inverse perspective mapping image. In *IEEE Intelligent Vehicles Symposium* (2019), pp. 317–323
14. L. Deng, M. Yang, H. Li, T. Li, B. Hu, C. Wang, Restricted deformable convolution-based road scene semantic segmentation using surround view cameras. IEEE Trans. Intell. Transp. Syst. **21**(10), 4350–4362 (2020)
[ADS][Crossref]
15. S. Yogamani, C. Hughes, J. Horgan, G. Sistu, P. Varley, D. O'Dea, M. Uricar, S. Milz, M. Simon, K. Amende, C. Witt, H. Rashed, S. Chennupati, S. Nayak, S. Mansoor, X. Perrotton, P. Perez, Woodscape: a multi-task, multi-camera fisheye dataset for autonomous driving, in *IEEE/CVF International Conference on Computer Vision* (2019), pp. 9308–9318
16. V. Kumar, C. Eising, C. Witt, S. Yogamani, Surround-view fisheye camera perception for automated driving: overview, survey & challenges. IEEE Trans. Intell. Transp. Syst. **24**(4), 3638–3659 (2023)
[Crossref]
17. R. Wood, Fish-eye views, and vision under water. Lond. Edinb. Dublin Philos. Mag. J. Sci. **12**(68), 159–162 (1906)
[Crossref]
18. W. Bond, A wide angle lens for cloud recording. Lond. Edinb. Dublin Philos. Mag. J. Sci. **44**(263), 999–1001 (1922)
[Crossref]
19. R. Feynman. *The Feynman Lectures on Physics (Commemorative Issue)* (Pearson Education, 2004)
20. R. Tsai, A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. IEEE J. Robot. Autom. **3**(4), 323–344 (1987)
[ADS][Crossref]
21. J. Weng, P. Cohen, M. Herniou, Camera calibration with distortion models and accuracy evaluation. IEEE Trans. Pattern Anal. Mach. Intell. **14**(10), 965–980 (1992)
[ADS][Crossref]
22. S. Lando. *Lectures on Generating Functions* (American Mathematical Society, 2003)
23. R. Hartley, A. Zisserman. *Multiple View Geometry in Computer Vision*. (Cambridge University Press, 2000)
24. L. Robert, C. Zeller, O. Faugeras, M. Hébert. *Applications of nonmetric vision to some visually guided robotics tasks*, in INRIA Research Report, RR-2584 (1995)
25. C. Loop, Z. Zhang, Computing rectifying homographies for stereo vision, in *IEEE Conference on Computer Vision and Pattern Recognition* (1999), pp. 125–131

26.
H. Hirschmuller, Stereo processing by semiglobal matching and mutual information. IEEE Trans. Pattern Anal. Mach. Intell. **30**(2), 328–341 (2008)
[ADS][Crossref][PubMed]

Footnotes

[1] This is like in daily life when one looks along a long straight road in the distance, the road boundaries that are parallel in the real-world will intersect at a vanishing point in her or his perspective view.

[2] In fact, even when distortion is taken into account, the concepts of the three coordinates systems are still valid and pertinent.

[3] The axes $O_\mathrm{c} - X_\mathrm{c}$ and $O_\mathrm{c} - Y_\mathrm{c}$ are chosen to be parallel actually with the axes of the *photosensitive plane* (refer to Sect. 1.2 in Chap. 1), i.e. the *physical image coordinates system* In contrast with the physical image coordinates system, the image coordinates system is of mathematical abstraction and has no physical instantiation. However, since coordinates in the image coordinates system and those in the physical image coordinates system differ only by some scaling factor on the whole, imagine that the axes $O_\mathrm{c} - X_\mathrm{c}$ and $O_\mathrm{c} - Y_\mathrm{c}$ are “virtually parallel” with the axes $O_\mathrm{I} - U$ and $O_\mathrm{I} - V$ of the image coordinates system.

[4] In fact, the 3D world counterparts of the image point form a line in the 3D world. To explain this obvious daily-life common sense theoretically, note that the two equations in (4.18) represent two planes respectively in the 3D world, so the solution set of (4.18) is right the intersection of the two planes and is a line obviously.

[5] The notation $\mathbf{e}_i$ denotes the *i*-th column vector of the identity matrix, so $\boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_1$ and $\boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_2$ represent the first and second column vectors of the matrix $\boldsymbol{R}_\mathrm{wc}$ respectively.

[6] Smart mobile phones, such as *HUAWEI* smart mobile phones and other kinds of smart mobile phones, are commonly available nowadays.

[7] “Tribute to Yan-An” is published on *China Daily*.

[8] It is worth noting that the demonstrated functionality of virtual file scanning may have already been incorporated into smart mobile phones at readers’ hands. So in practice, readers may not really need to take a camera photo and then perform inverse perspective mapping by themselves. Yet above presentation can promote readers’ understanding of the working mechanism of such interesting and useful functionality.

[9] To understand this point, just reflect on how to cut off the A4 paper area directly from the original camera image, which is in fact not so convenient to handle.

[10] To understand this point, think about what if two pixels of the original camera image are inversely projected to the same pixel of the normal view image and what if some pixel of the normal view image is not the inverse projection of any pixel of the original camera image.

[11] These are not actual image coordinates but virtual ones that exist in people’s imagination.

[12] Once the coefficients d_1, d_2, d_3, ... are treated as free coefficients, (4.35) can hence be treated as a pure linear form of these free coefficients, which tends to be more tractable than the original nonlinear form (4.34).

[13] The *inverse function theorem* [22] states given a function

$$B(t) = b_1 t + b_2 t^2 + b_3 t^3 + \cdots \quad (b_1 \neq 0),$$

then there exist functions

$$A(t) = a_1 t + a_2 t^2 + a_3 t^3 + \cdots$$

and

$$C(t) = c_1 t + c_2 t^2 + c_3 t^3 + \cdots$$

such that

$$A(B(t)) = t, \quad B(C(t)) = t.$$

Each of the functions $A(\cdot)$ and $C(\cdot)$ is a unique function possessing this property. The functions $A(\cdot)$ and $C(\cdot)$ are *left inverse* and *right inverse* respectively to the function $B(\cdot)$.

[14] As explained in Sect. 4.3.3, actual implementation of inverse perspective mapping does not follow the formalism of inverse perspective mapping literally. Similarly, actual implementation of distortion rectification does not follow the formalism of the inverse image distortion model literally, but still resorts to the formalism of the image distortion model and an interpolation practice like what is explained in Sect. 4.3.3.

[15] A human’s pair of eyes is a good biological example of binocular vision.

[16] See definition of the *skew-symmetric matrix* in Sect. D.2.1 in Appendix D.

[17] Suppose after image rectification, the camera intrinsic parameter $\beta = \alpha$ and the camera intrinsic parameter $\gamma = 0$.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_5

5. Camera Calibration

Hao Li[1] ✉
(1) Shanghai Jiao Tong University, Shanghai, China

5.1 Camera Intrinsic Calibration
5.1.1 Exploitation of Known Homography
5.1.2 Chessboard Pattern for Camera Intrinsic Calibration
5.1.3 Initial Estimate of Camera Intrinsic Parameters
5.2 Refined Estimate of Camera Intrinsic Parameters
5.2.1 Three-Dimensional Rotation and Rodrigues Formula
5.2.2 Lie Group Interpretation of Three-Dimensional Rotation
5.2.3 Estimate Refinement via Optimization
5.3 Image Distortion Calibration
5.4 Camera-X Calibration
5.4.1 Fundamental Matrix Calibration
5.4.2 Essential Matrix Calibration
5.4.3 Stereo Camera Calibration
5.4.4 Camera-Range Sensor Calibration
References

In practical applications, the camera extrinsic parameters tend to vary during operation of the computer vision system and may be on-line estimated or adjusted a posteriori, whereas the camera intrinsic parameters (including distortion coefficients) are normally time-invariant during operation of the computer vision system and can be calibrated a priori.

Besides, when multiple cameras form a fixed multi-vision system or when a camera and another kind of perceptive sensor form a fixed multi-sensor system, then the time-invariant spatial relationships among the multiple cameras or the multiple perceptive sensors can also be calibrated a priori.

5.1 Camera Intrinsic Calibration

A general philosophy for camera calibration is to take advantage of certain *control points* on certain specially-designed pattern to establish equations in terms of camera parameters, by solving which one can reveal camera parameters. In the spirit of such general philosophy, *Zhang* proposed in the late 1990s a milestone method for *camera intrinsic calibration*, i.e. calibration of camera intrinsic parameters [1], which treats corner points on a chessboard pattern as control points and takes advantage of them to subtly establish conveniently-solvable equations in terms of camera intrinsic .parameters

5.1.1 Exploitation of Known Homography

Consider a plane in the 3D world and a world coordinates system that is established with its $O_\mathrm{w} - X_\mathrm{w} - Y_\mathrm{w}$ plane located on the target plane, as illustrated in Fig. 4.9. Suppose the homography between the target plane and the image plane is known; for example, suppose the homography matrix $\boldsymbol{H}$ in (4.19), i.e.

$$\boldsymbol{H} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_1 \quad \boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_2 \quad \boldsymbol{T}_\mathrm{wc}] \equiv \boldsymbol{A}[\boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_1 \quad \boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_2 \quad \boldsymbol{T}_\mathrm{wc}]$$

is calibrated using the method presented in Sect. 4.3.2 in Chap. 4. In fact, as already explained in Sect. 4.3.2, the homography matrix has no influence at all on the plane-to-plane bijective relationship characterized by it if it changes only by a scaling factor. What can be calibrated or known is not the homography matrix $\boldsymbol{H}$ itself, but a scaled version of it namely $\lambda\boldsymbol{H}$ where the scaling factor λ is unknown. Fortunately, the unknown scaling factor λ does not influence following analysis at all.

Denote the homography matrix $\boldsymbol{H}$ in details as

$$\boldsymbol{H} \equiv [\boldsymbol{H}_{:,1} \quad \boldsymbol{H}_{:,2} \quad \boldsymbol{H}_{:,3}] \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

As supposed above, the holistic values of $\lambda\boldsymbol{H}_{:,1}$, $\lambda\boldsymbol{H}_{:,2}$, and $\lambda\boldsymbol{H}_{:,3}$ are known for some unknown scaling factor λ, namely

$$\lambda\boldsymbol{H}_{:,1} = \boldsymbol{H}^*_{:,1}, \quad \lambda\boldsymbol{H}_{:,2} = \boldsymbol{H}^*_{:,2}, \quad \lambda\boldsymbol{H}_{:,3} = \boldsymbol{H}^*_{:,3}$$

or

$$\begin{bmatrix} \lambda h_{11} \\ \lambda h_{21} \\ \lambda h_{31} \end{bmatrix} = \begin{bmatrix} h^*_{11} \\ h^*_{21} \\ h^*_{31} \end{bmatrix}, \quad \begin{bmatrix} \lambda h_{12} \\ \lambda h_{22} \\ \lambda h_{32} \end{bmatrix} = \begin{bmatrix} h^*_{12} \\ h^*_{22} \\ h^*_{32} \end{bmatrix}, \quad \begin{bmatrix} \lambda h_{13} \\ \lambda h_{23} \\ \lambda h_{33} \end{bmatrix} = \begin{bmatrix} h^*_{13} \\ h^*_{23} \\ h^*_{33} \end{bmatrix},$$

where

$$\boldsymbol{H}^*_{:,1} \equiv \begin{bmatrix} h^*_{11} \\ h^*_{21} \\ h^*_{31} \end{bmatrix}, \quad \boldsymbol{H}^*_{:,2} \equiv \begin{bmatrix} h^*_{12} \\ h^*_{22} \\ h^*_{32} \end{bmatrix}, \quad \boldsymbol{H}^*_{:,3} \equiv \begin{bmatrix} h^*_{13} \\ h^*_{23} \\ h^*_{33} \end{bmatrix}$$

are known for some unknown λ.

Note that

$$\boldsymbol{H}_{:,1} = \quad \boldsymbol{A}\boldsymbol{R}_\mathrm{wc}\boldsymbol{e}_1, \tag{5.1a}$$

$$\boldsymbol{H}_{:,2} = \quad \boldsymbol{A}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2, \tag{5.1b}$$

$$\boldsymbol{H}_{:,3} = \quad \boldsymbol{A}\boldsymbol{T}_{\text{wc}} \tag{5.1c}$$

and the camera intrinsic matrix

$$\boldsymbol{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

is invertible, so

$$\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1 = \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,1} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,1}, \tag{5.2a}$$

$$\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2 = \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,2} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,2}, \tag{5.2b}$$

$$\boldsymbol{T}_{\text{wc}} = \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,3} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,3}. \tag{5.2c}$$

By associating the two equalities

$$(\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1 = (\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2, \quad (\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2 = 0$$

with the first two equations of (5.2), one can establish two equations in terms of $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$:

$$\begin{aligned} \boldsymbol{H}^{*\mathrm{T}}_{:,1}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,1} = &\quad \lambda^2(\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1 = \lambda^2(\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2 \\ = &\quad \boldsymbol{H}^{*\mathrm{T}}_{:,2}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,2}, \end{aligned} \tag{5.3a}$$

$$\boldsymbol{H}^{*\mathrm{T}}_{:,1}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^*_{:,2} = \quad \lambda^2(\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_1)^{\mathrm{T}}\boldsymbol{R}_{\text{wc}}\boldsymbol{e}_2 = 0. \tag{5.3b}$$

Express the matrix $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ in terms of the camera intrinsic parameters α, β, γ, u_0, and v_0 as

$$\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & -\frac{\beta u_0-\gamma v_0}{\alpha^2\beta} \\ [9pt] -\frac{\gamma}{\alpha^2\beta} & \frac{1}{\beta^2}+\frac{\gamma^2}{\alpha^2\beta^2} & \frac{\gamma(\beta u_0-\gamma v_0)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} \\ [9pt] -\frac{\beta u_0-\gamma v_0}{\alpha^2\beta} & \frac{\gamma(\beta u_0-\gamma v_0)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} & 1+\frac{v_0^2}{\beta^2}+\frac{(\beta u_0-\gamma v_0)^2}{\alpha^2\beta^2} \end{bmatrix} \equiv \begin{bmatrix} b_1 & b_2 & b_4 \\ b_2 & b_3 & b_5 \\ b_4 & b_5 & b_6 \end{bmatrix}. \tag{5.4}$$

To facilitate formula derivation of (5.4), one can resort to *symbolic operation* that has been incorporated into software such as Matlab. Matlab code for computing the matrix $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ via symbolic operation is given as follows.

ComputeInvAtA.m

```
%% Compute A^-T A^-1
syms a b r u0 v0
A = [a, r, u0; ...
     0, b, v0; ...
     0, 0, 1];
B = simplify(inv(transpose(A))*inv(A));
fprintf('A ='); disp(A); fprintf('A^-T A^-1 ='); disp(B);
```

Transform (5.3) into homogeneous linear equations in terms of the vector

$$\boldsymbol{b} \equiv \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix}^{\mathrm{T}} \tag{5.5}$$

that is formed by concatenating the upper-triangular elements of the matrix $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$. For this purpose, take advantage of *matrix vectorization* and *Kronecker product* (see Sect. 2.1.3 in Chap. 2). Vectorize $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ as

$$\mathrm{vec}(\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \equiv \boldsymbol{P}_{\mathrm{b}}\boldsymbol{b},$$

where $\boldsymbol{P}_{\mathrm{b}}$ denotes the 9-by-6 matrix.

Vectorize the two equations of (5.3) respectively as

$$\begin{aligned} 0 = \quad & \mathrm{vec}(\boldsymbol{H}^{*\mathrm{T}}_{:,1}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^{*}_{:,1} - \boldsymbol{H}^{*\mathrm{T}}_{:,2}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^{*}_{:,2}) \\ = \quad & (\boldsymbol{H}^{*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1} - \boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,2})\mathrm{vec}(\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}) \\ = \quad & (\boldsymbol{H}^{*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1} - \boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,2})\boldsymbol{P}_{\mathrm{b}}\boldsymbol{b}, \\ 0 = \quad & \mathrm{vec}(\boldsymbol{H}^{*\mathrm{T}}_{:,1}\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{H}^{*}_{:,2}) = (\boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1})\mathrm{vec}(\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}) \\ = \quad & (\boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1})\boldsymbol{P}_{\mathrm{b}}\boldsymbol{b}, \end{aligned}$$

namely

$$\begin{bmatrix} (\boldsymbol{H}^{*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1} - \boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,2})\boldsymbol{P}_{\mathrm{b}} \\ (\boldsymbol{H}^{*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{*\mathrm{T}}_{:,1})\boldsymbol{P}_{\mathrm{b}} \end{bmatrix} \boldsymbol{b} = \boldsymbol{0}. \tag{5.6}$$

Therefore, given a configuration of known homography, i.e.

$$\lambda \boldsymbol{H} = \boldsymbol{H}^{*} \equiv [\boldsymbol{H}^{*}_{:,1} \quad \boldsymbol{H}^{*}_{:,2} \quad \boldsymbol{H}^{*}_{:,3}]$$

is known, then two homogeneous linear equations in terms of the vectorized $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ specified in (5.5) can be established, as described in (5.6).

If there are n configurations of known homography, i.e.

$$\begin{aligned} \lambda \boldsymbol{H}^{[1]} = \quad & \boldsymbol{H}^{[1]*} \equiv \left[\boldsymbol{H}^{[1]*}_{:,1} \quad \boldsymbol{H}^{[1]*}_{:,2} \quad \boldsymbol{H}^{[1]*}_{:,3}\right], \\ \lambda \boldsymbol{H}^{[2]} = \quad & \boldsymbol{H}^{[2]*} \equiv \left[\boldsymbol{H}^{[2]*}_{:,1} \quad \boldsymbol{H}^{[2]*}_{:,2} \quad \boldsymbol{H}^{[2]*}_{:,3}\right], \\ \cdots \quad & \cdots \\ \lambda \boldsymbol{H}^{[n]} = \quad & \boldsymbol{H}^{[n]*} \equiv \left[\boldsymbol{H}^{[n]*}_{:,1} \quad \boldsymbol{H}^{[n]*}_{:,2} \quad \boldsymbol{H}^{[n]*}_{:,3}\right] \end{aligned}$$

are known, then a homogeneous linear equation group of $2n$ equations in terms of the vectorized $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ can be established as

$$\left[\begin{array}{c} (\boldsymbol{H}^{[1]*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{[1]*\mathrm{T}}_{:,1} - \boldsymbol{H}^{[1]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[1]*\mathrm{T}}_{:,2})\boldsymbol{P}_{\mathrm{b}} \\ {[9pt]}(\boldsymbol{H}^{[1]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[1]*\mathrm{T}}_{:,1})\boldsymbol{P}_{\mathrm{b}} \\ {[9pt]}(\boldsymbol{H}^{[2]*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{[2]*\mathrm{T}}_{:,1} - \boldsymbol{H}^{[2]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[2]*\mathrm{T}}_{:,2})\boldsymbol{P}_{\mathrm{b}} \\ {[9pt]}(\boldsymbol{H}^{[2]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[2]*\mathrm{T}}_{:,1})\boldsymbol{P}_{\mathrm{b}} \\ {[5pt]}\cdots \quad \cdots \\ {[5pt]}(\boldsymbol{H}^{[n]*\mathrm{T}}_{:,1} \otimes \boldsymbol{H}^{[n]*\mathrm{T}}_{:,1} - \boldsymbol{H}^{[n]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[n]*\mathrm{T}}_{:,2})\boldsymbol{P}_{\mathrm{b}} \\ {[9pt]}(\boldsymbol{H}^{[n]*\mathrm{T}}_{:,2} \otimes \boldsymbol{H}^{[n]*\mathrm{T}}_{:,1})\boldsymbol{P}_{\mathrm{b}} \end{array}\right] \boldsymbol{b} = \boldsymbol{0}. \tag{5.7}$$

5.1.2 Chessboard Pattern for Camera Intrinsic Calibration

Given multiple configurations of known homography, the homogeneous linear equation (5.7) can be established, by solving which one can obtain $\lambda \boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$, i.e. certain scaled version of $\boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$ and then recover the camera intrinsic matrix $\boldsymbol{A}$ from $\lambda \boldsymbol{A}^{-\mathrm{T}}\boldsymbol{A}^{-1}$. Before further clarification of the process, a natural question comes first: how to (conveniently) obtain multiple configurations of known homography?

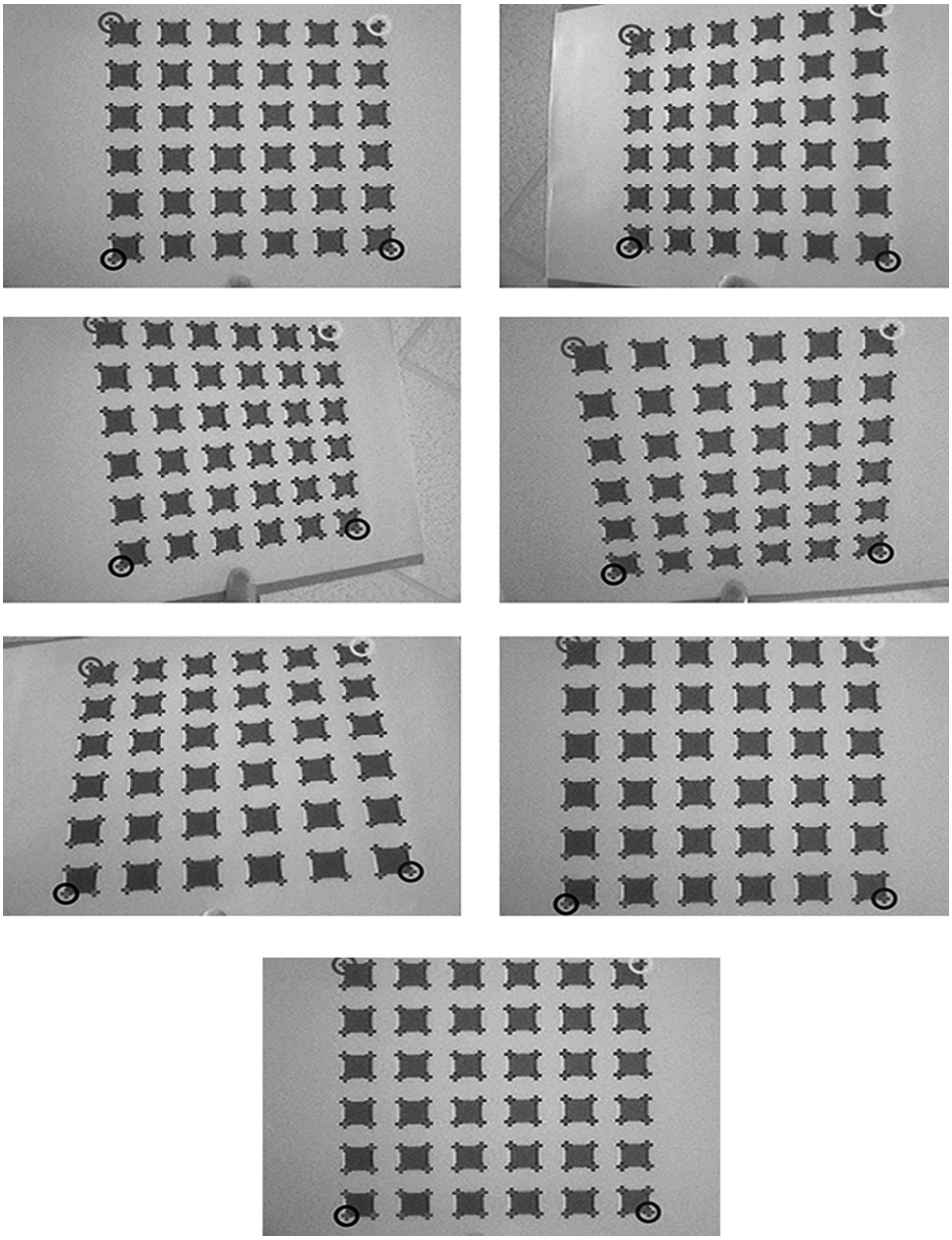


Fig. 5.1 Multiple chessboard poses for camera intrinsic calibration

To obtain multiple configurations of known homography, a conventional method adopted commonly nowadays is *Zhang*'s chessboard pattern method [1], as illustrated in Fig. 5.1. More specifically, print a chessboard pattern of known square sizes on a paper.[1] Then put the chessboard pattern at a number of different poses in front of the camera to-be-calibrated and capture corresponding images.

For each of the chessboard poses illustrated in Fig. 5.1, the world coordinates system $O_{w[i]} - X_{w[i]} - Y_{w[i]}$ is always attached to the chessboard plane, as illustrated in Fig. 5.2b, where the subscript "[i]" denotes the index of the chessboard pose among all chessboard poses. In the illustrated example, the square width is $w = 0.02$ m. There are $n = 12$ rows and $n = 12$ columns of control points (namely corner points of black squares), the world coordinates of which are constantly

$$\left\{\left[x_w^{r,c} \quad y_w^{r,c}\right]^{\mathrm{T}} \equiv \left[r \times w \quad c \times w\right]^{\mathrm{T}} \middle| r = 1, \ldots, n; c = 1, \ldots, n\right\}$$

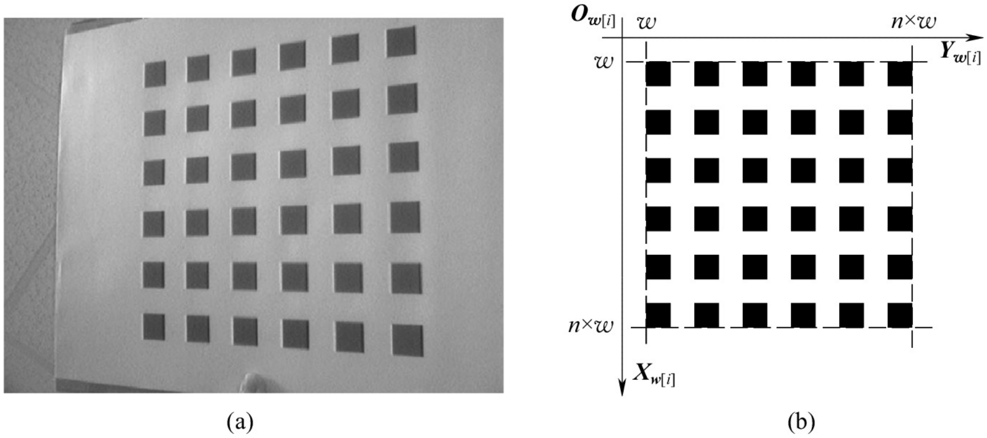


Fig. 5.2 Chessboard pattern **a** a chessboard pose; **b** world coordinates system attached to the chessboard plane

in the world coordinates system $O_{w[i]} - X_{w[i]} - Y_{w[i]}$. In other words, the world coordinates of the control points are known a priori.

Control points in the captured image corresponding to the chessboard pose areextracted and registered automatically. Once the image coordinates of the control points are known in such way, the homography associated with the chessboard pose is calibrated using the method presented in Sect. 4.3.2 in Chap. 4 and a configuration of known homography is obtained. For multiple chessboard poses, multiple configurations of known homography can be obtained in similar way and then a homogeneous linear equation (5.7) can be established.

Matlab code for automatic extraction and registration of control points on the chessboard pattern is given as follows. The sub-function **ConvexHull2D.m** is given in Sect. 3.4.1 in Chap. 3.

ExtractChessboardControlPoints.m

```
%% Extract and register chessboard control points automatically
% f : a struct of captured images of the chessboard poses
% nG: number of rows / columns of control points
% wG: black square width
function fo = ExtractChessboardControlPoints(f, nG, wG)
    fo = f; [H, W] = size(f(1).img);
    for k = 1:length(f)
        [ptH, imgH] = CornerHarris(f(k).img);
        i = find(ptH(1,:)>4 & ptH(1,:)<W-4 & ptH(2,:)>4 & ptH
    (2,:)<H-4);
        u = ptH(1,i); v = ptH(2,i); % Omit boundary
        [ch, cs] = ConvexHull2D([u;v]); % Convex hull of control
     points
        i = find(cs>-0.7); ch = ch(:,i); % Four array corners
        uc = [0, W, W, 0]; vc = [0, 0, H, H]; ic = zeros(1,4);
        for i2 = 1:4 % Match four corners of the chessboard
    array
            r = (ch(1,:)-uc(i2)).^2+(ch(2,:)-vc(i2)).^2;
            ic(i2) = find(r==min(r));
        end
        uc = ch(1,ic); vc = ch(2,ic);
        xc = [1, nG, nG, 1]*wG; yc = [1, 1, nG, nG]*wG;
        xG = reshape(repmat((1:nG)’,1,nG),1,[])*wG;
        yG = reshape(repmat(1:nG,nG,1),1,[])*wG;
        Hm = CalibrateHomography(uc, vc, xc, yc);
        [uG, vG] = HomographyXY2UV(Hm, xG, yG);
        for i2 = 1:length(uG) % Register chessboard control
    points
            d = abs(u-uG(i2))+abs(v-vG(i2)); i = find(d==min(d))
    ;
            uG(i2) = u(i); vG(i2) = v(i);
        end
        fo(k).uG=uG; fo(k).vG=vG; fo(k).xG=xG; fo(k).yG=yG;
        fo(k).uc=uc; fo(k).vc=vc; fo(k).xc=xc; fo(k).yc=yc;
    end
end
```

Once the homogeneous linear equation (5.7) is established according to multiple configurations of known homography, solve it and denote its solution as

$$\lambda \boldsymbol{b} \equiv [\lambda b_1 \quad \lambda b_2 \quad \lambda b_3 \quad \lambda b_4 \quad \lambda b_5 \quad \lambda b_6]^{\mathrm{T}} = \boldsymbol{b}^* \equiv [b_1^* \quad b_2^* \quad b_3^* \quad b_4^* \quad b_5^* \quad b_6^*]^{\mathrm{T}}, \qquad (5.8)$$

where $\boldsymbol{b}^*$ is known and λ is certain unknown scaling factor.

In Sect. 5.1.3 the author will present how to obtain an initial estimate of camera intrinsic parameters, and in Sect. 5.2 the author will further present how to obtain a refined estimate of camera intrinsic parameters via optimization.

5.1.3 Initial Estimate of Camera Intrinsic Parameters

Suppose the solution $\lambda \boldsymbol{b} = \boldsymbol{b}^*$ specified in (5.8) is known. Consider (5.4) and establish six equations in terms of the camera intrinsic parameters α, β, γ, u_0, and v_0 as

$$\frac{1}{\alpha^2} = \frac{b_1^*}{\lambda}, \tag{5.9a}$$

$$-\frac{\gamma}{\alpha^2\beta} = \frac{b_2^*}{\lambda}, \tag{5.9b}$$

$$\frac{1}{\beta^2} + \frac{\gamma^2}{\alpha^2\beta^2} = \frac{b_3^*}{\lambda}, \tag{5.9c}$$

$$-\frac{\beta u_0 - \gamma v_0}{\alpha^2\beta} = \frac{b_4^*}{\lambda}, \tag{5.9d}$$

$$\frac{\gamma(\beta u_0 - \gamma v_0)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} = \frac{b_5^*}{\lambda}, \tag{5.9e}$$

$$1 + \frac{v_0^2}{\beta^2} + \frac{(\beta u_0 - \gamma v_0)^2}{\alpha^2\beta^2} = \frac{b_6^*}{\lambda}. \tag{5.9f}$$

Divide the second equation of (5.9) by the first equation of (5.9) and obtain

$$\frac{\gamma}{\beta} = -\frac{b_2^*}{b_1^*}. \tag{5.10}$$

Substitute the first equation of (5.9) and (5.10) into the third equation of (5.9) and obtain

$$\frac{b_1^*}{\lambda}\left(-\frac{b_2^*}{b_1^*}\right)^2 + \frac{1}{\beta^2} = \frac{b_3^*}{\lambda} \Longleftrightarrow \beta = \sqrt{\frac{\lambda b_1^*}{b_1^* b_3^* - b_2^{*2}}}. \tag{5.11}$$

Substitute the fourth equation of (5.9), (5.10), and (5.11) into the fifth equation of (5.9) and obtain

$$\frac{b_2^*}{b_1^*}\frac{b_4^*}{\lambda} - v_0\frac{b_1^* b_3^* - b_2^{*2}}{\lambda b_1^*} = \frac{b_5^*}{\lambda} \Longleftrightarrow v_0 = \frac{b_2^* b_4^* - b_1^* b_5^*}{b_1^* b_3^* - b_2^{*2}}. \tag{5.12}$$

Substitute above results into the sixth equation of (5.9) and obtain

$$\frac{\lambda}{b_1^*}\left(\frac{b_4^*}{\lambda}\right)^2 + \frac{v_0(b_2^* b_4^* - b_1^* b_5^*)}{\lambda b_1^*} + 1 = \frac{b_6^*}{\lambda} \Longleftrightarrow \lambda = b_6^* - \frac{b_4^{*2} + v_0(b_2^* b_4^* - b_1^* b_5^*)}{b_1^*}. \tag{5.13}$$

Then from the first equation of (5.9) one can compute

$$\alpha = \sqrt{\frac{\lambda}{b_1^*}}, \tag{5.14}$$

from (5.11) one can compute β, and further from (5.10) one can compute γ. Finally from the fourth equation of (5.9) one can compute

$$u_0 = \frac{v_0\gamma}{\beta} - \frac{b_4^*\alpha^2}{\lambda}. \tag{5.15}$$

In summary, the camera intrinsic parameters are estimated as

$$v_0 = \frac{b_2^* b_4^* - b_1^* b_5^*}{b_1^* b_3^* - b_2^{*2}}, \tag{5.16a}$$

$$\begin{aligned} \lambda &= b_6^* - \frac{b_4^{*2} + v_0(b_2^* b_4^* - b_1^* b_5^*)}{b_1^*}, \\ \alpha &= \sqrt{\frac{\lambda}{b_1^*}}, \end{aligned} \tag{5.16b}$$

$$\beta = \sqrt{\frac{\lambda b_1^*}{b_1^* b_3^* - b_2^{*2}}}, \tag{5.16c}$$

$$\gamma = -\frac{b_2^*}{b_1^*}\beta, \tag{5.16d}$$

$$u_0 = \frac{v_0\gamma}{\beta} - \frac{b_4^*\alpha^2}{\lambda}. \tag{5.16e}$$

Matlab code for initial estimation of the camera intrinsic parameters (or the camera intrinsic matrix) is given as follows.

InitialEstimateCameraIntrinsic.m

```
%% Initial estimate of camera intrinsic parameters
function camA = InitialEstimateCameraIntrinsic(f)
    Pb = [1,0,0,0,0,0; 0,1,0,0,0,0; 0,0,0,1,0,0; ...
             0,1,0,0,0,0; 0,0,1,0,0,0; 0,0,0,0,1,0; ...
             0,0,0,1,0,0; 0,0,0,0,1,0; 0,0,0,0,0,1]; L = [];
    for k = 1:length(f)
        H = CalibrateHomography(f(k).uG, f(k).vG, f(k).xG, f(k).
    yG);
        H1 = transpose(H(:,1)); H2 = transpose(H(:,2));
        Lt = [kron(H1,H1)-kron(H2,H2); kron(H2,H1)]*Pb; L = [L;
    Lt];
    end
    vB = num2cell(SolveHomogeneousLinearEquation(L));
    [b1, b2, b3, b4, b5, b6] = vB{:};
    v0 = (b2*b4-b1*b5)/(b1*b3-b2^2);
    s = b6 - (b4^2+v0*(b2*b4-b1*b5))/b1;
    a = sqrt(s/b1);
    b = sqrt(s*b1/(b1*b3-b2^2));
    r = (-b2/b1)*b;
    u0 = v0*r/b - b4*a^2/s;
    camA = [a, r, u0; 0, b, v0; 0, 0, 1];
end
```

5.2 Refined Estimate of Camera Intrinsic Parameters

One can further refine the initial estimate of camera intrinsic parameters given in (5.16). For this purpose, some basic knowledge of the three-dimensional rotation matrix and the Rodrigues formula needs to be clarified.

5.2.1 Three-Dimensional Rotation and Rodrigues Formula

Consider the three-dimensional rotation around a generic axis represented by a normalized vector $\bar{\boldsymbol{w}} \equiv \begin{bmatrix} \bar{w}_x & \bar{w}_y & \bar{w}_z \end{bmatrix}^{\mathrm{T}}$, as illustrated in Fig. 5.3. Suppose the rotation speed is v; then given an arbitrary 3D point $\boldsymbol{x}$, its rotation around the axis $\bar{\boldsymbol{w}}$ can be described by the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x} = v(\bar{\boldsymbol{w}} \times \boldsymbol{x}) = v\boldsymbol{S}_{\bar{\boldsymbol{w}}}\boldsymbol{x},$$

where $\boldsymbol{S}_{\bar{\boldsymbol{w}}}$ denotes the *skew-symmetric matrix* [2] of $\bar{\boldsymbol{w}}$

$$\boldsymbol{S}_{\bar{\boldsymbol{w}}} = \begin{bmatrix} 0 & -\bar{w}_z & \bar{w}_y \\ \bar{w}_z & 0 & -\bar{w}_x \\ -\bar{w}_y & \bar{w}_x & 0 \end{bmatrix}.$$

Solve above differential equation and obtain

$$\boldsymbol{x} = \mathrm{e}^{v\boldsymbol{S}_{\bar{\boldsymbol{w}}}t}\boldsymbol{x}_0,$$

which implies that the **rotation matrix** is

$$\boldsymbol{R} = \mathrm{e}^{v\boldsymbol{S}_{\bar{\boldsymbol{w}}}t} = \mathrm{e}^{\boldsymbol{S}_{\bar{\boldsymbol{w}}}\theta}, \tag{5.17}$$

where $\theta = vt$ is the rotation angle.

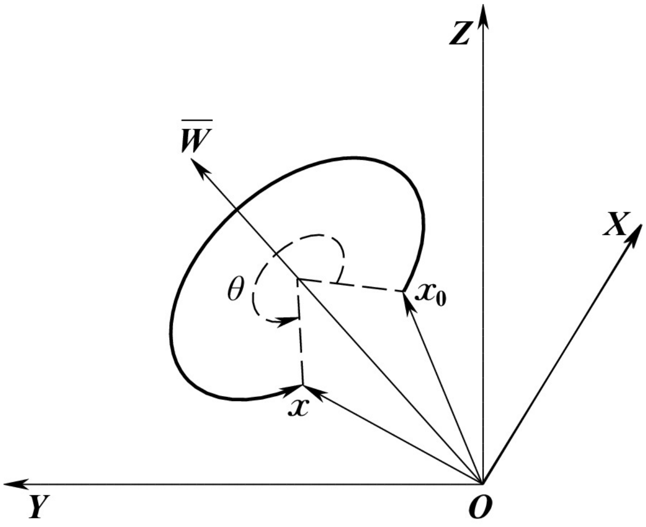


Fig. 5.3 Three-dimensional rotation

Define the **rotation vector** $\boldsymbol{w}$ as

$$\boldsymbol{w} = \bar{\boldsymbol{w}}\theta = [\bar{w}_x \quad \bar{w}_y \quad \bar{w}_z]^{\mathrm{T}}\theta \tag{5.18}$$

and the rotation matrix formalism (5.17) becomes

$$\boldsymbol{R} = \mathrm{e}^{\boldsymbol{S}_{\bar{\boldsymbol{w}}}\theta} = \mathrm{e}^{\boldsymbol{S}_{\bar{\boldsymbol{w}}\theta}} = \mathrm{e}^{\boldsymbol{S}_{\boldsymbol{w}}}, \tag{5.19}$$

which conveys the relationship between the rotation vector $\boldsymbol{w}$ and the rotation matrix $\boldsymbol{R}$.

According to (5.19), one can interpret the physical meaning of a generic rotation vector $\boldsymbol{w}$ as follows: the normalized version of $\boldsymbol{w}$, i.e. $\bar{\boldsymbol{w}} = \boldsymbol{w}/\|\boldsymbol{w}\|_2$ represents the rotation axis, whereas the norm of $\boldsymbol{w}$, i.e. $\theta = \|\boldsymbol{w}\|_2$ represents the rotation angle.

From (5.19) one further derives

$$\begin{aligned}
\boldsymbol{R}(\boldsymbol{w}) = {} & \mathrm{e}^{\boldsymbol{S}_{\boldsymbol{w}}} = \sum_{k=0}^{\infty} \frac{\boldsymbol{S}_{\boldsymbol{w}}^{k}}{k!} = \boldsymbol{I} + \sum_{k=0}^{\infty} \frac{\boldsymbol{S}_{\boldsymbol{w}}^{2k+1}}{(2k+1)!} + \sum_{k=0}^{\infty} \frac{\boldsymbol{S}_{\boldsymbol{w}}^{2k+2}}{(2k+2)!} \\
= {} & \boldsymbol{I} + \boldsymbol{S}_{\boldsymbol{w}} \sum_{k=0}^{\infty} (-1)^{k} \frac{\|\boldsymbol{w}\|_2^{2k}}{(2k+1)!} + \boldsymbol{S}_{\boldsymbol{w}}^{2} \sum_{k=0}^{\infty} (-1)^{k} \frac{\|\boldsymbol{w}\|_2^{2k}}{(2k+2)!} \\
= {} & \boldsymbol{I} + \boldsymbol{S}_{\bar{\boldsymbol{w}}} \sum_{k=0}^{\infty} (-1)^{k} \frac{\|\boldsymbol{w}\|_2^{2k+1}}{(2k+1)!} + \boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2} \sum_{k=0}^{\infty} (-1)^{k} \frac{\|\boldsymbol{w}\|_2^{2k+2}}{(2k+2)!} \\
= {} & \boldsymbol{I} + \boldsymbol{S}_{\bar{\boldsymbol{w}}} \sin \|\boldsymbol{w}\|_2 + \boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2}(1 - \cos \|\boldsymbol{w}\|_2) = \boldsymbol{I} + \boldsymbol{S}_{\bar{\boldsymbol{w}}} \sin \theta + \boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2}(1 - \cos \theta).
\end{aligned} \tag{5.20}$$

In derivation of (5.20), the following equalities

$$\boldsymbol{S}_{\boldsymbol{w}}^{2} = \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}} - \|\boldsymbol{w}\|_2^2 \boldsymbol{I}, \quad \boldsymbol{S}_{\boldsymbol{w}}^{3} = -\|\boldsymbol{w}\|_2^2 \boldsymbol{S}_{\boldsymbol{w}}$$

are used. Besides, the rotation matrix is expressed as a function $\boldsymbol{R}(\boldsymbol{w})$ explicitly in terms of the rotation vector $\boldsymbol{w}$ to highlight the relationship between them. The mapping described by the rotation function $\boldsymbol{R}(\boldsymbol{w})$ is *surjective* but not *injective*: every rotation matrix has corresponding rotation vectors, and different rotation vectors can correspond to the same rotation matrix. If one constrains the domain of $\boldsymbol{w}$ to certain range say

$$0 \leqslant \|\boldsymbol{w}\| < 2\pi,$$

then the mapping described by the rotation function $\boldsymbol{R}(\boldsymbol{w})$ becomes *bijective*. When the rotation function $\boldsymbol{R}(\boldsymbol{w})$ is bijective, denote the inverse rotation function as $\boldsymbol{R}^{-1}(\cdot)$ namely $\boldsymbol{w} = \boldsymbol{R}^{-1}(\boldsymbol{R})$.

The formula given in (5.20) is called the **Rodrigues formula** [2].[3] Matlab code for implementing the Rodrigues formula to compute the rotation matrix of a rotation vector is given as follows.

RotationVectorToMatrix.m

```
%% Rotation Vector to Rotation Matrix
function R = RotationVectorToMatrix(w)
    a = sqrt(transpose(w)*w); % Rotation angle
    if (a<10*eps) R = eye(3); return; end % Negligible rotation
    wn = w/a; % Normalized rotation vector
    Swn = [0, -wn(3), wn(2); ...
          wn(3), 0, -wn(1); ...
          -wn(2), wn(1), 0];
    R = eye(3) + Swn*sin(a) + Swn^2*(1-cos(a)); % Rodrigues
    formula
end
```

Given an arbitrary rotation vector, one now knows how to obtain its corresponding rotation matrix via the Rodrigues formula. On the other hand, given an arbitrary rotation matrix

$$\boldsymbol{R}=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{5.21}$$

how can we obtain its corresponding rotation vector?

Express the Rodrigues formula (5.20) in details as

$$\boldsymbol{R}(\boldsymbol{w})=\begin{bmatrix} 1+(\bar{w}_x^2-1)(1-c_\theta) & -\bar{w}_z s_\theta+\bar{w}_x\bar{w}_y(1-c_\theta) & \bar{w}_y s_\theta+\bar{w}_x\bar{w}_z(1-c_\theta) \\ \bar{w}_z s_\theta+\bar{w}_y\bar{w}_x(1-c_\theta) & 1+(\bar{w}_y^2-1)(1-c_\theta) & -\bar{w}_x s_\theta+\bar{w}_y\bar{w}_z(1-c_\theta) \\ -\bar{w}_y s_\theta+\bar{w}_z\bar{w}_x(1-c_\theta) & \bar{w}_x s_\theta+\bar{w}_z\bar{w}_y(1-c_\theta) & 1+(\bar{w}_z^2-1)(1-c_\theta) \end{bmatrix}, \tag{5.22}$$

where $s_\theta \equiv \sin\theta$ and $c_\theta \equiv \cos\theta$.

A method is to directly compare rotation matrix elements between (5.21) and (5.22). First, compare corresponding diagonal elements

$$1+(\bar{w}_x^2-1)(1-c_\theta) = \quad r_{11}, \tag{5.23a}$$

$$1+(\bar{w}_y^2-1)(1-c_\theta) = \quad r_{22}, \tag{5.23b}$$

$$1+(\bar{w}_z^2-1)(1-c_\theta) = \quad r_{33}. \tag{5.23c}$$

Then summate the three equations of (5.23) into

$$3+(\bar{w}_x^2+\bar{w}_y^2+\bar{w}_z^2-3)(1-c_\theta)=1+2c_\theta=r_{11}+r_{22}+r_{33}$$

from which one can solve the rotation angle θ as

$$\theta = \arccos\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right) = \arccos\left(\frac{\mathrm{tr}(\boldsymbol{R})-1}{2}\right). \tag{5.24}$$

Substitute (5.24) back into (5.23) and obtain

$$\bar{w}_x = \quad \pm\sqrt{\frac{1+r_{11}-r_{22}-r_{33}}{3-r_{11}-r_{22}-r_{33}}}, \tag{5.25a}$$

$$\bar{w}_y = \quad \pm\sqrt{\frac{1-r_{11}+r_{22}-r_{33}}{3-r_{11}-r_{22}-r_{33}}}, \tag{5.25b}$$

$$\bar{w}_z = \quad \pm\sqrt{\frac{1-r_{11}-r_{22}+r_{33}}{3-r_{11}-r_{22}-r_{33}}}. \tag{5.25c}$$

The signs of $\bar{w}_x$, $\bar{w}_y$, and $\bar{w}_z$ in (5.25) can be determined by comparing non-diagonal elements between (5.21) and (5.22). Finally, one can compute the rotation vector $\boldsymbol{w}$ via (5.18). Matlab code for computing the rotation vector of a rotation matrix via this method is given as follows.

DirectRotationMatrixToVector.m

```
%% Direct Rotation Matrix to Rotation Vector
function w = DirectRotationMatrixToVector(R)
    vecR = num2cell(reshape(R,1,[]));
    [r11, r21, r31, r12, r22, r32, r13, r23, r33] = vecR{:};
    ca = (r11+r22+r33-1)/2; a = acos(ca);
    wx = sqrt(max(0,(1+r11-r22-r33)/(3-r11-r22-r33)));
    wy = sqrt(max(0,(1-r11+r22-r33)/(3-r11-r22-r33)));
    wz = sqrt(max(0,(1-r11-r22+r33)/(3-r11-r22-r33)));
    dMin = 10000;
    for i = 0:7
        i1 = floor(i/4); i2 = floor((i-i1*4)/2); i3 = i-i1*4-i2
    *2;
        wn = [(-1)^i1*wx; (-1)^i2*wy; (-1)^i3*wz];
        Swn = [0, -wn(3), wn(2); wn(3), 0, -wn(1); -wn(2), wn(1)
    , 0];
        Rw = eye(3) + Swn*sin(a) + Swn^2*(1-cos(a));
        d = sum(sum(abs(Rw-R))); if (d<dMin) w = wn*a; dMin = d;
     end
    end
end
```

Another method which might be more elegant is to take advantage of the fact that the (normalized) rotation vector is actually an eigenvector of the rotation matrix and its associated eigenvalue is 1, namely

$$\boldsymbol{R}\boldsymbol{w} = \boldsymbol{w} \quad \text{or} \quad \mathbf{R}\bar{\mathbf{w}} = \bar{\mathbf{w}},$$

the physical meaning of which can be interpreted as that a point on the rotation axis is still itself after rotation. Transform above equality into a homogeneous linear equation in terms of the normalized rotation vector $\bar{\boldsymbol{w}}$

$$(\boldsymbol{R}-\boldsymbol{I})\bar{\boldsymbol{w}}=\mathbf{0}. \tag{5.26}$$

Solve the homogeneous linear equation (5.26) to obtain the normalized rotation vector $\bar{\boldsymbol{w}}$ and substitute $\bar{\boldsymbol{w}}$ into the Rodrigues formula (5.20)

$$\boldsymbol{I}+\boldsymbol{S}_{\bar{\boldsymbol{w}}}\sin\theta+\boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2}(1-\cos\theta)=\boldsymbol{R}. \tag{5.27}$$

Vectorize both sides of (5.27) to establish a linear equation in terms of $(s_\theta, c_\theta)\equiv(\sin\theta, \cos\theta)$ as

$$\begin{bmatrix}\mathrm{vec}(\boldsymbol{S}_{\bar{\boldsymbol{w}}}) & -\mathrm{vec}(\boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2})\end{bmatrix}\begin{bmatrix}s_\theta\\ c_\theta\end{bmatrix}=\mathrm{vec}(\boldsymbol{R}-\boldsymbol{I}-\boldsymbol{S}_{\bar{\boldsymbol{w}}}^{2}). \tag{5.28}$$

Solve the linear equation (5.28) to obtain the solution of (s_θ, c_θ). Due to this or that kind of error in practical applications, the given rotation matrix $\boldsymbol{R}$ usually cannot strictly satisfy the orthonormal condition and hence the solved solution of (s_θ, c_θ) usually cannot strictly satisfy the normalization condition either. Normalize (s_θ, c_θ) as

$$\sin\theta=\frac{s_\theta}{\sqrt{s_\theta^2+c_\theta^2}},\quad \cos\theta=\frac{c_\theta}{\sqrt{s_\theta^2+c_\theta^2}},$$

from which one can recover the rotation angle θ. Since the normalized rotation vector $\bar{\boldsymbol{w}}$ has already been obtained by solving the homogeneous linear equation (5.26), one can finally compute the rotation vector $\boldsymbol{w}$ via (5.18). Matlab code for exploiting the Rodrigues formula to compute the rotation vector of a rotation matrix is given as follows.

RotationMatrixToVector.m

```
%% Rotation Matrix to Rotation Vector
function w = RotationMatrixToVector(R)
    Rm = R - eye(3); % note (R-I)*w = 0
    if (trace(abs(Rm))<10*eps) w = [0;0;0]; return; end
    wn = SolveHomogeneousLinearEquation(Rm); % Normalized
    rotation vector
    Swn = [0, -wn(3), wn(2); ...
          wn(3), 0, -wn(1); ...
          -wn(2), wn(1), 0];
    Swn2 = Swn^2;
    A = [reshape(Swn,[],1), -reshape(Swn2,[],1)];
    B = reshape(R-eye(3)-Swn2,[],1);
    sc = A\B; sc = sc/sqrt(transpose(sc)*sc); % [sin(a); cos(a)]
    a = atan2(sc(1), sc(2)); % Rotation angle
    w = wn*a;
end
```

5.2.2 Lie Group Interpretation of Three-Dimensional Rotation

Interpretation of three-dimensional rotation and derivation of the Rodrigues formula (5.20) presented in Sect. 5.2.1 resort to physical intuition of rotation laws.

Readers may wonder whether one could interpret three-dimensional rotation and derive the Rodrigues formula without relying on physical intuition but purely in abstract mathematical way. The answer is indeed yes.

For the purpose, consider the *special orthogonal group*

$$\boldsymbol{SO}(3) \equiv \{\boldsymbol{R} | \boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I}, \det\,(\boldsymbol{R}) = 1\} \tag{5.29}$$

and treat it as a **Lie group**.[4] A generic $\boldsymbol{R} \in \boldsymbol{SO}(3)$ is associated with a matrix $\boldsymbol{W}$ via the *matrix exponential function*

$$\boldsymbol{R} = \boldsymbol{I} + \boldsymbol{W} + \frac{\boldsymbol{W}^2}{2!} + \frac{\boldsymbol{W}^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{\boldsymbol{W}^k}{k!} \equiv \mathrm{e}^{\boldsymbol{W}} \tag{5.30}$$

and inversely via the *matrix logarithm function*

$$\boldsymbol{W} = (\boldsymbol{R}-\boldsymbol{I}) - \frac{(\boldsymbol{R}-\boldsymbol{I})^2}{2} + \frac{(\boldsymbol{R}-\boldsymbol{I})^3}{3} - \frac{(\boldsymbol{R}-\boldsymbol{I})^4}{4} + \cdots = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{(\boldsymbol{R}-\boldsymbol{I})^k}{k}. \tag{5.31}$$

It is worth noting that (5.31) holds when $\boldsymbol{R}$ is close to the identity matrix $\boldsymbol{I}$ (specifically, $\|\boldsymbol{R} - \boldsymbol{I}\| < 1$). If $\boldsymbol{R}$ is not close enough to the identity matrix $\boldsymbol{I}$, one can first compute the square root of $\boldsymbol{R}$, denoted as $\sqrt{\boldsymbol{R}}$. Substitute $\sqrt{\boldsymbol{R}}$ for $\boldsymbol{R}$ in (5.31) and compute a $\boldsymbol{W}_2$. Then one has $\boldsymbol{W} = 2\boldsymbol{W}_2$ such that $\mathrm{e}^{\boldsymbol{W}} = \boldsymbol{R}$. If $\sqrt{\boldsymbol{R}}$ is still not close enough to the identity matrix $\boldsymbol{I}$, one can further compute the square root of $\sqrt{\boldsymbol{R}}$ and apply (5.31) in similar way to compute $\boldsymbol{W}$. Here, the author puts aside such details of how to apply (5.31) flexibly. After all, the key point of (5.30) and (5.31) is to highlight the bijective relationship between $\boldsymbol{R}$ and $\boldsymbol{W}$ (within certain range).

Consider a generic differential of the generic $\boldsymbol{R}$

$$\mathrm{d}\boldsymbol{R} = \mathrm{d}(\mathrm{e}^{\boldsymbol{W}}) = \mathrm{e}^{\boldsymbol{W}}\mathrm{d}\boldsymbol{W} = \boldsymbol{R}\mathrm{d}\boldsymbol{W} \tag{5.32}$$

and generic differentials of both sides of the equality $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I}$, i.e.

$$\begin{aligned} \mathrm{d}(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}) = &\quad (\mathrm{d}\boldsymbol{R})^{\mathrm{T}}\boldsymbol{R} + \boldsymbol{R}^{\mathrm{T}}(\mathrm{d}\boldsymbol{R}) = (\boldsymbol{R}\mathrm{d}\boldsymbol{W})^{\mathrm{T}}\boldsymbol{R} + \boldsymbol{R}^{\mathrm{T}}(\boldsymbol{R}\mathrm{d}\boldsymbol{W}) \\ = &\quad (\mathrm{d}\boldsymbol{W})^{\mathrm{T}} + \mathrm{d}\boldsymbol{W} = \mathrm{d}\boldsymbol{I} = \boldsymbol{0}. \end{aligned} \tag{5.33}$$

The equality (5.33) conveys that the differentials on the manifold $\boldsymbol{SO}(3)$ are all *skew-symmetric*. Further consider an arbitrary integral path of differentials from the identity matrix $\boldsymbol{I} = \mathrm{e}^{\boldsymbol{0}}$ to $\boldsymbol{R} = \mathrm{e}^{\boldsymbol{W}}$

$$\boldsymbol{W} = \boldsymbol{0} + \int_{\text{certain path}} \mathrm{d}\boldsymbol{W} \tag{5.34}$$

which implies $\boldsymbol{W}$ must be skew-symmetric as well. Besides, given a generic skew-symmetric matrix $\boldsymbol{W}$, there is a corresponding $\boldsymbol{R} = \mathrm{e}^{\boldsymbol{W}} \in \boldsymbol{SO}(3)$ because

$$\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \mathrm{e}^{\boldsymbol{W}^{\mathrm{T}}}\mathrm{e}^{\boldsymbol{W}} = \mathrm{e}^{-\boldsymbol{W}}\mathrm{e}^{\boldsymbol{W}} = \mathrm{e}^{\boldsymbol{W}(-1+1)} = \mathrm{e}^{\boldsymbol{0}} = \boldsymbol{I} \tag{5.35}$$

and

$$\begin{aligned}\det\left(s\boldsymbol{I}-\mathrm{e}^{\boldsymbol{W}}\right)= & \det\left[s\boldsymbol{I}-\left(\boldsymbol{I}+\bar{\boldsymbol{W}}\sin\theta+\bar{\boldsymbol{W}}^{2}(1-\cos\theta)\right)\right]\\ = & (s-1)(s^{2}-2\cos\theta s+1)\Rightarrow\det\left(\mathrm{e}^{\boldsymbol{W}}\right)=1.\end{aligned} \tag{5.36}$$

Therefore, one has established bijective relationship between the special orthogonal group $\boldsymbol{SO}(3)$ and the skew-symmetric matrix space $\varOmega_{\boldsymbol{W}}\equiv\{\boldsymbol{W}|\boldsymbol{W}^{\mathrm{T}}+\boldsymbol{W}=\boldsymbol{0}\}$. Since there is bijective relationship between the skew-symmetric matrix space $\Omega_{\boldsymbol{W}}$ and the three-dimensional space, one has also established bijective relationship between the special orthogonal group $\boldsymbol{SO}(3)$ and the three-dimensional space (namely that of rotation vectors introduced in Sect. 5.2.1), yet without resorting to physical intuition of rotation laws.

It is worth noting that in derivation of (5.35), the special associative law $\mathrm{e}^{\boldsymbol{W}t_1}\mathrm{e}^{\boldsymbol{W}t_2}=\mathrm{e}^{\boldsymbol{W}(t_1+t_2)}$ is used. Readers had better not take this special associative law for granted. It needs to be proved: $\forall t_1,t_2$, define the function $\phi(t)=\mathrm{e}^{\boldsymbol{W}(t_1-t)}\mathrm{e}^{\boldsymbol{W}(t+t_2)}$. One has

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\phi(t)=\tfrac{\mathrm{d}}{\mathrm{d}t}\mathrm{e}^{\boldsymbol{W}(t_1-t)}\mathrm{e}^{\boldsymbol{W}(t+t_2)}+\mathrm{e}^{\boldsymbol{W}(t_1-t)}\tfrac{\mathrm{d}}{\mathrm{d}t}\mathrm{e}^{\boldsymbol{W}(t+t_2)}=-\boldsymbol{W}\phi(t)+\boldsymbol{W}\phi(t)=\boldsymbol{0},$$

which implies that the function $\phi(t)$ is a constant matrix regardless of t. So

$$\mathrm{e}^{\boldsymbol{W}t_1}\mathrm{e}^{\boldsymbol{W}t_2}=\phi(0)=\phi(t_1)=\mathrm{e}^{\boldsymbol{W}(t_1+t_2)}.$$

In (5.36), $\boldsymbol{W}=\bar{\boldsymbol{W}}\theta$, where $\bar{\boldsymbol{W}}$ is the normalized version of $\boldsymbol{W}$. The equality

$$\mathrm{e}^{\boldsymbol{W}}=\boldsymbol{I}+\bar{\boldsymbol{W}}\sin\theta+\bar{\boldsymbol{W}}^{2}(1-\cos\theta)$$

namely the Rodrigues formula (5.20) is used.[5] For computation of the characteristic polynomial $\det\left(s\boldsymbol{I}-\mathrm{e}^{\boldsymbol{W}}\right)$, one can resort to *symbolic operation* that has been incorporated into software such as Matlab. Matlab code for computing the characteristic polynomial $\det\left(s\boldsymbol{I}-\mathrm{e}^{\boldsymbol{W}}\right)$ via symbolic operation is given as follows.

ComputeEigExpW.m

```
%% Characteristic polynomial of exp(W)
syms x y a s
z = sqrt(1-x^2-y^2); Wn = [0,-z,y;z,0,-x;-y,x,0];
eW = eye(3) + Wn*sin(a) + Wn^2*(1-cos(a));
eWc = simplify(collect(det(s*eye(3)-eW),s));
simplify(transpose(eW)*eW), eWc
```

Equation (5.32) characterizes the *tangent space* of the special orthogonal group $\boldsymbol{SO}(3)$ at $\boldsymbol{R}$, which is *isomorphic* to the tangent space of $\boldsymbol{SO}(3)$ at the group identity $\mathrm{e}^{\boldsymbol{0}}=\boldsymbol{I}$. More specifically, if one shifts the special orthogonal group $\boldsymbol{SO}(3)$ by the group operation $\boldsymbol{R}^{-1}$, then the tangent space becomes identical to the tangent space of $\boldsymbol{SO}(3)$ at the group identity. Such kind of *isomorphism* can be interpreted from the differential geometry perspective as follows: differentials on the manifold $\boldsymbol{SO}(3)$ can be characterized by the differentials within the infinitesimal neighbourhood of the group identity $\mathrm{e}^{\mathbf{0}} = \boldsymbol{I}$.

When $\boldsymbol{R} = \boldsymbol{I}$ namely at the group identity, (5.32) is reduced to

$$\mathrm{d}\boldsymbol{R}|_{\boldsymbol{R}=\boldsymbol{I}} = \boldsymbol{R}\mathrm{d}\boldsymbol{W} = \mathrm{d}\boldsymbol{W}.$$

So the tangent space at the group identity is right the skew-symmetric matrix space $\Omega_{\boldsymbol{W}}$, which forms the **Lie algebra** of the Lie group $\boldsymbol{SO}(3)$. The bijective relationship between the special orthogonal group $\boldsymbol{SO}(3)$ and the skew-symmetric matrix space $\Omega_{\boldsymbol{W}}$ can also be treated as the bijective relationship between the Lie group $\boldsymbol{SO}(3)$ and its Lie algebra .[6] In the Lie algebra $\Omega_{\boldsymbol{W}}$, definition of the multiplication just follows the *Poisson bracket*: Given $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \Omega_{\boldsymbol{W}}$, their multiplication is

$$[\boldsymbol{W}_1, \boldsymbol{W}_2] = \boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{W}_2\boldsymbol{W}_1 = \boldsymbol{S}_{\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}} \tag{5.37}$$

where notations $\boldsymbol{S}_{\{\cdot\}}$ and $\boldsymbol{S}^{[-1]}_{\{\cdot\}}$ are defined in Sect, D.2.1 in Appendix D. Proof of the equality

$$\boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{W}_2\boldsymbol{W}_1 = \boldsymbol{S}_{\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}}$$

in (5.37) is given as follows.

Proof Note that the vector cross product is a well-known binary operation that satisfies the three conditions concerning Lie algebra, namely the distributive law, the anticommutative law, and the Jacobi identity. So

$$\begin{aligned}
& \boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{W}_2\boldsymbol{W}_1 = \boldsymbol{S}_{\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}} \\
\Leftrightarrow\ & \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times \boldsymbol{v}) - \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{v}) = (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}) \times \boldsymbol{v} \quad (\forall \boldsymbol{v} \in \mathbf{R}^3) \\
\Leftrightarrow\ & \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times \boldsymbol{v}) - \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{v}) - (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}) \times \boldsymbol{v} = \mathbf{0} \\
& (\text{anticommutative law of the cross product}) \\
\Leftrightarrow\ & \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times \boldsymbol{v}) + \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2} \times (\boldsymbol{v} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1}) + \boldsymbol{v} \times (\boldsymbol{S}^{[-1]}_{\boldsymbol{W}_1} \times \boldsymbol{S}^{[-1]}_{\boldsymbol{W}_2}) = \mathbf{0}. \\
& (\text{Jacobi identity of the cross product})
\end{aligned}$$

The proof is done. □

For a group $\boldsymbol{G}$, the *commutator* [4] is an important concept in the sense that the *commutator subgroup* is the smallest normal subgroup such that the *quotient group* of the group $\boldsymbol{G}$ by the subgroup is Abelian, i.e. commutative. Given two generic $\boldsymbol{R}_1 = \mathrm{e}^{\boldsymbol{W}_1}, \boldsymbol{R}_2 = \mathrm{e}^{\boldsymbol{W}_2} \in \boldsymbol{SO}(3)$ in the infinitesimal neighbourhood of the group identity $\mathrm{e}^{\mathbf{0}} = \boldsymbol{I}$, their commutator is

$$
\begin{aligned}
\boldsymbol{R}_1\boldsymbol{R}_2\boldsymbol{R}_1^{-1}\boldsymbol{R}_2^{-1} &= \mathrm{e}^{\boldsymbol{W}_1}\mathrm{e}^{\boldsymbol{W}_2}\mathrm{e}^{-\boldsymbol{W}_1}\mathrm{e}^{-\boldsymbol{W}_2} \\
&= \left(\boldsymbol{I}+\boldsymbol{W}_1+\frac{\boldsymbol{W}_1^2}{2!}+\cdots\right)\left(\boldsymbol{I}+\boldsymbol{W}_2+\frac{\boldsymbol{W}_2^2}{2!}+\cdots\right) \\
&\quad \left(\boldsymbol{I}-\boldsymbol{W}_1+\frac{\boldsymbol{W}_1^2}{2!}+\cdots\right)\left(\boldsymbol{I}-\boldsymbol{W}_2+\frac{\boldsymbol{W}_2^2}{2!}+\cdots\right) \\
&= \boldsymbol{I}+\boldsymbol{W}_1\boldsymbol{W}_2-\boldsymbol{W}_2\boldsymbol{W}_1+o(\boldsymbol{W}_1^2,\boldsymbol{W}_2^2) \\
&= \boldsymbol{I}+[\boldsymbol{W}_1,\boldsymbol{W}_2]+o(\boldsymbol{W}_1^2,\boldsymbol{W}_2^2)\approx \mathrm{e}^{[\boldsymbol{W}_1,\boldsymbol{W}_2]}.
\end{aligned}
$$

As conveyed by the above analysis, the Poisson bracket characterizes local linearization of “noncommutativeness” of the Lie group $\boldsymbol{SO}(3)$.

5.2.3 Estimate Refinement via Optimization

Multiple chessboard poses namely multiple configurations of homography, such as illustrated in Fig. 5.1, are used for camera intrinsic calibration—as already mentioned, this well-known method is *Zhang*’s chessboard pattern method [1]—Suppose there are totally *n* chessboard poses. For the *k*-th chessboard pose, denote the rotation matrix and the translation vector between its attached world coordinates system and the camera coordinates system as $\boldsymbol{R}_{\mathrm{wc}}^{[k]}$ and $\boldsymbol{T}_{\mathrm{wc}}^{[k]}$ respectively.

In the chessboard pattern method, corners of black squares are chosen as *control points* for sake of extraction convenience. In fact, the *n* configurations of known homography $\boldsymbol{H}^{[1]*}$, $\boldsymbol{H}^{[2]*}, \ldots, \boldsymbol{H}^{[n]*}$, which enables initial estimation of camera intrinsic parameters via (5.7) and (5.16), are obtained right with the help of these control points.

For the *k*-th chessboard pose, denote the world coordinates of the control points in its attached world coordinates system as

$$
\boldsymbol{P}_{\mathrm{w},1}^{[k]} \equiv \begin{bmatrix} x_{\mathrm{w},1}^{[k]} \\ \lbrack 6pt\rbrack y_{\mathrm{w},1}^{[k]} \\ 0 \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{w},2}^{[k]} \equiv \begin{bmatrix} x_{\mathrm{w},2}^{[k]} \\ \lbrack 6pt\rbrack y_{\mathrm{w},2}^{[k]} \\ 0 \end{bmatrix}, \ldots, \boldsymbol{P}_{\mathrm{w},m}^{[k]} \equiv \begin{bmatrix} x_{\mathrm{w},m}^{[k]} \\ \lbrack 9pt\rbrack y_{\mathrm{w},m}^{[k]} \\ 0 \end{bmatrix}
$$

$$
\boldsymbol{P}_{\mathrm{i},1}^{[k]} \equiv \begin{bmatrix} u_1^{[k]} \\ \lbrack 6pt\rbrack v_1^{[k]} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},2}^{[k]} \equiv \begin{bmatrix} u_2^{[k]} \\ \lbrack 6pt\rbrack v_2^{[k]} \end{bmatrix}, \ldots, \boldsymbol{P}_{\mathrm{i},m}^{[k]} \equiv \begin{bmatrix} u_m^{[k]} \\ \lbrack 6pt\rbrack v_m^{[k]} \end{bmatrix},
$$

where m denotes the number of control points and $k \in \{1, \ldots, n\}$. All these world coordinates and image coordinates of the control points in the multiple chessboard poses can be known a priori, either in manual way or in automatic way.

According to the pinhole camera model (4.7), the camera intrinsic matrix

$$
\boldsymbol{A} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}
$$

and the camera extrinsic parameters associated with the n chessboard poses namely the rotation matrices and the translation vectors

$$\boldsymbol{R}_{\mathrm{wc}}^{[1]}, \quad \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]}, \quad \boldsymbol{T}_{\mathrm{wc}}^{[2]}, \ldots, \boldsymbol{R}_{\mathrm{wc}}^{[n]}, \quad \boldsymbol{T}_{\mathrm{wc}}^{[n]}$$

should satisfy

$$\lambda \begin{bmatrix} \boldsymbol{u}_1^{[1]} \\ \boldsymbol{v}_1^{[1]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[1]}\ \boldsymbol{T}_{\mathrm{wc}}^{[1]}\right] \begin{bmatrix} \boldsymbol{x}_{\mathrm{w},1}^{[1]} \\ \boldsymbol{y}_{\mathrm{w},1}^{[1]} \\ 0 \\ 1 \end{bmatrix}, \ \ldots, \ \lambda \begin{bmatrix} \boldsymbol{u}_{\boldsymbol{m}}^{[1]} \\ \boldsymbol{v}_{\boldsymbol{m}}^{[1]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[1]}\ \boldsymbol{T}_{\mathrm{wc}}^{[1]}\right] \begin{bmatrix} \boldsymbol{x}_{\mathrm{w},\boldsymbol{m}}^{[1]} \\ \boldsymbol{y}_{\mathrm{w},\boldsymbol{m}}^{[1]} \\ 0 \\ 1 \end{bmatrix},$$

or compactly

$$\begin{aligned}
\lambda \boldsymbol{P}_{\mathrm{i},1}^{[1]A} &= \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[1]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[1]}\right] \boldsymbol{P}_{\mathrm{w},1}^{[1]A}, \ldots, \lambda \boldsymbol{P}_{\mathrm{i},m}^{[1]A} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[1]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[1]}\right] \boldsymbol{P}_{\mathrm{w},m}^{[1]A}, \\
\lambda \boldsymbol{P}_{\mathrm{i},1}^{[2]A} &= \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[2]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[2]}\right] \boldsymbol{P}_{\mathrm{w},1}^{[2]A}, \ldots, \quad \lambda \boldsymbol{P}_{\mathrm{i},m}^{[2]A} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[2]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[2]}\right] \boldsymbol{P}_{\mathrm{w},m}^{[2]A}, \\
\cdots \quad & \quad \cdots \qquad\qquad\qquad\qquad \cdots \qquad\qquad \cdots \quad \cdots \\
\lambda \boldsymbol{P}_{\mathrm{i},1}^{[n]A} &= \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[n]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[n]}\right] \boldsymbol{P}_{\mathrm{w},1}^{[n]A}, \ldots, \lambda \boldsymbol{P}_{\mathrm{i},m}^{[n]A} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[n]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[n]}\right] \boldsymbol{P}_{\mathrm{w},m}^{[n]A}.
\end{aligned}$$

One can establish a group of nonlinear equations in terms of $\{\boldsymbol{A}, \boldsymbol{R}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{R}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$:

$$\frac{\begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \\ \boldsymbol{e}_2^{\mathrm{T}} \end{bmatrix} \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[k]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}\right] \boldsymbol{P}_{\mathrm{w},l}^{[k]A}}{\boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[k]} \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}\right] \boldsymbol{P}_{\mathrm{w},l}^{[k]A}} = \boldsymbol{P}_{\mathrm{i},l}^{[k]}, \tag{5.38}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$.

Denote the camera intrinsic parameters in vector form as

$$\boldsymbol{v}_{\boldsymbol{A}} \equiv \begin{bmatrix} \alpha & \beta & \gamma & u_0 & v_0 \end{bmatrix}^{\mathrm{T}}$$

and abuse the notation $\boldsymbol{A}(\cdot)$ to denote the function that maps $\boldsymbol{v}_{\boldsymbol{A}}$ to its associated camera intrinsic matrix

$$\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}}) = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

According to Sect. 5.2.1, each rotation matrix $\boldsymbol{R}_{\mathrm{wc}}^{[k]}$ can be characterized by a rotation vector. Abuse the notation $\boldsymbol{R}(\cdot)$ to denote the rotation function that maps a rotation vector to its associated rotation matrix via the Rodrigues formula (5.20) and then denote

$$\boldsymbol{R}_{\mathrm{wc}}^{[1]} = \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[1]}), \quad \boldsymbol{R}_{\mathrm{wc}}^{[2]} = \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[2]}), \ldots, \boldsymbol{R}_{\mathrm{wc}}^{[n]} = \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[n]}).$$

Therefore, the group of nonlinear equations (5.38) can be formalized equivalently as a group of nonlinear equations in terms of free variables $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$:

$$\frac{\begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \\ \boldsymbol{e}_2^{\mathrm{T}} \end{bmatrix} \boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}}) \left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right] \boldsymbol{P}_{\text{w},l}^{[k]A}}{\boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}}) \left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right] \boldsymbol{P}_{\text{w},l}^{[k]A}} = \boldsymbol{P}_{\text{i},l}^{[k]}, \tag{5.39}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$.

To solve the group of nonlinear equations (5.39), an initial estimate of $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$ is needed. The initial estimate of $\boldsymbol{v}_{\boldsymbol{A}}$ can be obtained via (5.16). For the k-th chessboard pose, to obtain the initial estimate of $\boldsymbol{w}_{\text{wc}}^{[k]}$ and $\boldsymbol{T}_{\text{wc}}^{[k]}$, resort to(5.2):

$$\begin{aligned}
\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_1 &= \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}^{[k]} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}^{[k]*}, \\
\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_2 &= \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}^{[k]} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}^{[k]*}, \\
\boldsymbol{T}_{\text{wc}}^{[k]} &= \quad \boldsymbol{A}^{-1}\boldsymbol{H}_{:,3}^{[k]} = \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,3}^{[k]*},
\end{aligned}$$

where $\boldsymbol{H}^{[k]*}$ is already known. Since both $\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_1$ and $\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_2$ satisfy the normalization condition, one has

$$\begin{aligned}
\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_1 &= \quad \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}^{[k]*}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}^{[k]*}\|_2}, \\
\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_2 &= \quad \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}^{[k]*}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}^{[k]*}\|_2}, \\
\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_3 &= \quad (\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_1) \times (\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_2), \\
\boldsymbol{w}_{\text{wc}}^{[k]} &= \quad \boldsymbol{R}^{-1}(\boldsymbol{R}_{\text{wc}}^{[k]}) = \boldsymbol{R}^{-1}\left(\left[\boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_1 \quad \boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_2 \quad \boldsymbol{R}_{\text{wc}}^{[k]}\boldsymbol{e}_3\right]\right),
\end{aligned} \tag{5.40a}$$

$$\begin{aligned}
\lambda &= \quad \sqrt{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}^{[k]*}\|_2 \cdot \|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}^{[k]*}\|_2}, \\
\boldsymbol{T}_{\text{wc}}^{[k]} &= \quad \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,3}^{[k]*},
\end{aligned} \tag{5.40b}$$

where $\boldsymbol{R}^{-1}(\cdot)$ denotes the function that maps a (quasi) rotation matrix to its associated rotation vector via the method presented in Sect. 5.2.1.

Once an initial estimate of $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$ is available, solve the group of nonlinear equations (5.39) iteratively in the spirit of numeric optimization. Methods of solving nonlinear equations are presented in Sect. F.3 in Appendix F.

Matlab code for initial estimation of camera full parameters, i.e. camera intrinsic and extrinsic parameters $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$ is given as follows. In the demonstrated Matlab code, all the camera parameters are concatenated into a large vector (called the camera parameter vector) that is to be refined iteratively. The orders of camera parameters in the vector do not matter, only if they are fixed and registered well for convenient use.

$$\lambda\begin{bmatrix} u_1^{[2]} \\ v_1^{[2]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[2]}\ \boldsymbol{T}_{\mathrm{wc}}^{[2]}\right]\begin{bmatrix} x_{\mathrm{w},1}^{[2]} \\ y_{\mathrm{w},1}^{[2]} \\ 0 \\ 1 \end{bmatrix},\ \ldots,\ \lambda\begin{bmatrix} u_m^{[2]} \\ v_m^{[2]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[2]}\ \boldsymbol{T}_{\mathrm{wc}}^{[2]}\right]\begin{bmatrix} x_{\mathrm{w},m}^{[2]} \\ y_{\mathrm{w},m}^{[2]} \\ 0 \\ 1 \end{bmatrix},$$

$$\cdots\quad\cdots\qquad\qquad\cdots\qquad\qquad\cdots\quad\cdots$$

$$\lambda\begin{bmatrix} u_1^{[n]} \\ v_1^{[n]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[n]}\ \boldsymbol{T}_{\mathrm{wc}}^{[n]}\right]\begin{bmatrix} x_{\mathrm{w},1}^{[n]} \\ y_{\mathrm{w},1}^{[n]} \\ 0 \\ 1 \end{bmatrix},\ \ldots,\ \lambda\begin{bmatrix} u_m^{[n]} \\ v_m^{[n]} \\ 1 \end{bmatrix} = \boldsymbol{A}\left[\boldsymbol{R}_{\mathrm{wc}}^{[n]}\ \boldsymbol{T}_{\mathrm{wc}}^{[n]}\right]\begin{bmatrix} x_{\mathrm{w},m}^{[n]} \\ y_{\mathrm{w},m}^{[n]} \\ 0 \\ 1 \end{bmatrix},$$

Matlab code for batch projection of the world coordinates (of control points) to the image coordinates is given as follows.

InitialEstimateCameraFullParameters.m

```
%% Initial estimate of camera full parameters
% camV = [[a; b; r; u0; v0]; wR1; T1; wR2; T2; ...; wRn; Tn]
% [a; b; r; u0; v0]: camera intrinsic parameters
% wR{1,2,...,n}: rotation vectors associated with chessboard
    poses
% T{1,2,...,n} : translation vectors associated with chessboard
    poses
% wG: world coordinates batch of control points
% iG: image coordinates batch of control points
function [camV, iG, wG] = InitialEstimateCameraFullParameters(f)
    n = length(f); camV = zeros(6*n+5,1);
    nG = length(f(1).uG); iG = zeros(2*nG,n); wG = zeros(3*nG,n)
    ;
    camA = InitialEstimateCameraIntrinsic(f); invA = inv(camA);
    camV(1:5) = [camA(1,1);camA(2,2);camA(1,2);camA(1,3);camA
    (2,3)];
    for k = 1:n
        uG = reshape(f(k).uG,1,[]); vG = reshape(f(k).vG,1,[]);
        xG = reshape(f(k).xG,1,[]); yG = reshape(f(k).yG,1,[]);
        H = CalibrateHomography(uG, vG, xG, yG);
        R1 = invA*H(:,1); nR1 = sqrt(transpose(R1)*R1); R1 = R1/
    nR1;
        R2 = invA*H(:,2); nR2 = sqrt(transpose(R2)*R2); R2 = R2/
    nR2;
        T = invA*H(:,3)/sqrt(nR1*nR2);
        if (T(3)<0) R1 = -R1; R2 = -R2; T = -T; end
        R3 = [0, -R1(3), R1(2); R1(3), 0, -R1(1); -R1(2), R1(1),
     0]*R2;
        camV(6*k:6*k+2) = RotationMatrixToVector([R1, R2, R3]);
        camV(6*k+3:6*k+5) = T;
        iG(:,k) = reshape([uG; vG],[],1);
        wG(:,k) = reshape([xG; yG; zeros(1,nG)],[],1);
    end
    iG = reshape(iG,[],1);
end
```

Once the parametric nonlinear function **CameraProjectionBatch.m** is defined, one can resort to the Matlab code **SolveParametricNonlinearEquation.m** presented in Sect. F.3 in Appendix F to refine camera intrinsic parameters via the *Levenberg-Marquardt method* [5, 6]. Complete Matlab code for camera intrinsic calibration is given as follows.

CameraProjectionBatch.m

```
%% Camera Projection Batch
% camV = [[a; b; r; u0; v0]; wR1; T1; wR2; T2; ...; wRn; Tn]
% [a; b; r; u0; v0]: camera intrinsic parameters
% wR{1,2,...,n}: rotation vectors batch
% T{1,2,...,n} : translation vectors batch
% wG: given world coordinates batch
% iG: projected image coordinates batch
function iG = CameraProjectionBatch(camV, wG)
    n = size(wG,2); nG = size(wG,1)/3; iG = zeros(2*nG,n);
    camA = [camV(1), camV(3), camV(4); 0, camV(2), camV(5); 0,
    0, 1];
    for k = 1:n
        wAug = [wG(1:3:end,k)';wG(2:3:end,k)';wG(3:3:end,k)';
    ones(1,nG)];
        Rwc = RotationVectorToMatrix(camV(6*k:6*k+2)); %
    Rotation matrix
        Twc = camV(6*k+3:6*k+5); % Translation vector
        iAug = camA*[Rwc,Twc]*wAug; % Perspective mapping
        uG = iAug(1,:)./iAug(3,:); vG = iAug(2,:)./iAug(3,:);
        iG(:,k) = reshape([uG;vG],[],1);
    end
    iG = reshape(iG,[],1);
end
```

5.3 Image Distortion Calibration

For image distortion calibration, refer to the distortion-considered camera model (4.38)

$$
\begin{aligned}
\begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R}_\mathrm{wc} & \boldsymbol{T}_\mathrm{wc} \end{bmatrix} \begin{bmatrix} x_\mathrm{w} \\ y_\mathrm{w} \\ z_\mathrm{w} \\ 1 \end{bmatrix}, \\
\lambda \begin{bmatrix} \hat{u} \\ \hat{v} \\ 1 \end{bmatrix} &= \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix}, \\
\begin{bmatrix} u \\ v \end{bmatrix} &= \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix} \left[1 + d_1 \left(\frac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2} \right) + d_2 \left(\frac{x_\mathrm{c}^2 + y_\mathrm{c}^2}{z_\mathrm{c}^2} \right)^2 \right],
\end{aligned}
$$

i.e.

$$\begin{aligned}
\boldsymbol{P}_{\text{c}} &= [\boldsymbol{R}_{\text{wc}} \quad \boldsymbol{T}_{\text{wc}}]\boldsymbol{P}_{\text{w}}^{A},\\
\lambda\hat{\boldsymbol{P}}_{\text{i}}^{A} &= \boldsymbol{A}\boldsymbol{P}_{\text{c}},\\
s_{\boldsymbol{P}_{\text{c}}} &= 1 + d_1\left(\boldsymbol{P}_{\text{c}}^{\text{T}}\frac{\boldsymbol{e}_1\boldsymbol{e}_1^{\text{T}}+\boldsymbol{e}_2\boldsymbol{e}_2^{\text{T}}}{\boldsymbol{P}_{\text{c}}^{\text{T}}\boldsymbol{e}_3\boldsymbol{e}_3^{\text{T}}\boldsymbol{P}_{\text{c}}}\boldsymbol{P}_{\text{c}}\right) + d_2\left(\boldsymbol{P}_{\text{c}}^{\text{T}}\frac{\boldsymbol{e}_1\boldsymbol{e}_1^{\text{T}}+\boldsymbol{e}_2\boldsymbol{e}_2^{\text{T}}}{\boldsymbol{P}_{\text{c}}^{\text{T}}\boldsymbol{e}_3\boldsymbol{e}_3^{\text{T}}\boldsymbol{P}_{\text{c}}}\boldsymbol{P}_{\text{c}}\right)^2,\\
\boldsymbol{P}_{\text{i}} &= \boldsymbol{P}_{\text{i},0} + (\hat{\boldsymbol{P}}_{\text{i}} - \boldsymbol{P}_{\text{i},0})s_{\boldsymbol{P}_{\text{c}}},
\end{aligned}$$

where only the first and second distortion parameters are taken into account, because they are rather enough to characterize image distortion in many practical applications. Even if more distortion parameters are taken into account, the spirit of the calibration method presented below still applies. So considering only the first and second distortion parameters has no essential influence on clarifying the calibration method.

Incorporate the distortion parameters d_1 and d_2 into the set of camera full parameters to enlarge it as $\{\boldsymbol{v}_{\boldsymbol{A}}, d_1, d_2, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$. Besides, adapt the group of nonlinear equations (5.39) accordingly as

$$\begin{aligned}
&\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \left(\frac{\begin{bmatrix} \boldsymbol{e}_1^{\text{T}} \\ \boldsymbol{e}_2^{\text{T}} \end{bmatrix}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}})\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}}{\boldsymbol{e}_3^{\text{T}}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}})\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}} - \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \right)\\
&\left[1 + d_1\frac{\left(\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}\right)^{\text{T}}(\boldsymbol{e}_1\boldsymbol{e}_1^{\text{T}}+\boldsymbol{e}_2\boldsymbol{e}_2^{\text{T}})\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}}{\left(\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}\right)^{\text{T}}\boldsymbol{e}_3\boldsymbol{e}_3^{\text{T}}\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}} + \right.\\
&\left. d_2\left(\frac{\left(\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}\right)^{\text{T}}(\boldsymbol{e}_1\boldsymbol{e}_1^{\text{T}}+\boldsymbol{e}_2\boldsymbol{e}_2^{\text{T}})\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}}{\left(\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}\right)^{\text{T}}\boldsymbol{e}_3\boldsymbol{e}_3^{\text{T}}\left[\boldsymbol{R}(\boldsymbol{w}_{\text{wc}}^{[k]}) \quad \boldsymbol{T}_{\text{wc}}^{[k]}\right]\boldsymbol{P}_{\text{w},l}^{[k]A}}\right)^2\right]\\
= \quad &\boldsymbol{P}_{\text{i},l}^{[k]},
\end{aligned} \tag{5.41}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$.

In fact, for expression conciseness, one can represent the dazzling left side of (5.41) compactly as a parametric nonlinear function

$$\boldsymbol{P}_{\text{i}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{P}_{\text{w}}), \tag{5.42}$$

where $\boldsymbol{x}$ denotes the camera parameter vector namely

$$\boldsymbol{x} \equiv \begin{bmatrix} \boldsymbol{v}_{\boldsymbol{A}} \\ d_1 \\ d_2 \\ \boldsymbol{w}_{\mathrm{wc}}^{[1]} \\ \boldsymbol{T}_{\mathrm{wc}}^{[1]} \\ \vdots \\ \boldsymbol{w}_{\mathrm{wc}}^{[n]} \\ \boldsymbol{T}_{\mathrm{wc}}^{[n]} \end{bmatrix}$$

and the function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{P}_{\mathrm{w}})$ itself follows the procedures given by (4.38) to compute the function output $\boldsymbol{P}_{\mathrm{i}}$ from the function input $\boldsymbol{P}_{\mathrm{w}}$. Then the nonlinear equation group (5.41) is formalized compactly as

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{P}_{\mathrm{w},l}^{[k]A}) = \boldsymbol{P}_{\mathrm{i},l}^{[k]}, \tag{5.43}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$.

The initial estimate of $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$ is obtained using the method described previously in Sect. 5.2, whereas the initial estimate of $\{d_1, d_2\}$ is obtained in the following way: Use the initial estimate of $\{\boldsymbol{v}_{\boldsymbol{A}}, \boldsymbol{w}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$ to compute image projections of control points, treat such image projections as "ideal" image coordinates, and establish a linear equation group in terms of d_1 and d_2. More specifically, compute

$$\begin{aligned} \boldsymbol{P}_{\mathrm{c},l}^{[k]} &= \begin{bmatrix} \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]} \end{bmatrix} \boldsymbol{P}_{\mathrm{w},l}^{[k]A}, \\ \begin{bmatrix} \hat{u}_l^{[k]} \\ \hat{v}_l^{[k]} \end{bmatrix} &= \frac{\begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \\ \boldsymbol{e}_2^{\mathrm{T}} \end{bmatrix} \boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}}) \boldsymbol{P}_{\mathrm{c},l}^{[k]}}{\boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}}) \boldsymbol{P}_{\mathrm{c},l}^{[k]}}, \\ a_l^{[k]} &= \boldsymbol{P}_{\mathrm{c},l}^{[k]T} \frac{\boldsymbol{e}_1 \boldsymbol{e}_1^{\mathrm{T}} + \boldsymbol{e}_2 \boldsymbol{e}_2^{\mathrm{T}}}{\boldsymbol{P}_{\mathrm{c},l}^{[k]T} \boldsymbol{e}_3 \boldsymbol{e}_3^{\mathrm{T}} \boldsymbol{P}_{\mathrm{c},l}^{[k]}} \boldsymbol{P}_{\mathrm{c},l}^{[k]}, \end{aligned}$$

and establish the linear equation group

$$\begin{aligned} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} + \left(\begin{bmatrix} \hat{u}_l^{[k]} \\ \hat{v}_l^{[k]} \end{bmatrix} - \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \right) (1 + d_1 a_l^{[k]} + d_2 a_l^{[k]2}) &= \begin{bmatrix} u_l^{[k]} \\ v_l^{[k]} \end{bmatrix} \\ \Longleftrightarrow \begin{bmatrix} a_l^{[k]}(\hat{u}_l^{[k]} - u_0) & a_l^{[k]2}(\hat{u}_l^{[k]} - u_0) \\ a_l^{[k]}(\hat{v}_l^{[k]} - v_0) & a_l^{[k]2}(\hat{v}_l^{[k]} - v_0) \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} &= \begin{bmatrix} u_l^{[k]} - \hat{u}_l^{[k]} \\ v_l^{[k]} - \hat{v}_l^{[k]} \end{bmatrix}, \end{aligned} \tag{5.44}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$. Then solve (5.44) to obtain the initial estimate of $\{d_1, d_2\}$.

Matlab code for initial estimation of camera full parameters (including distortion parameters) $\{\boldsymbol{v}_{\boldsymbol{A}}, d_1, d_2, \boldsymbol{w}_{\text{wc}}^{[1]}, \boldsymbol{T}_{\text{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\text{wc}}^{[n]}, \boldsymbol{T}_{\text{wc}}^{[n]}\}$ is given as follows. In the demonstrated Matlab code, all the camera parameters are also concatenated into a large vector namely the camera parameter vector that is to be refined iteratively.

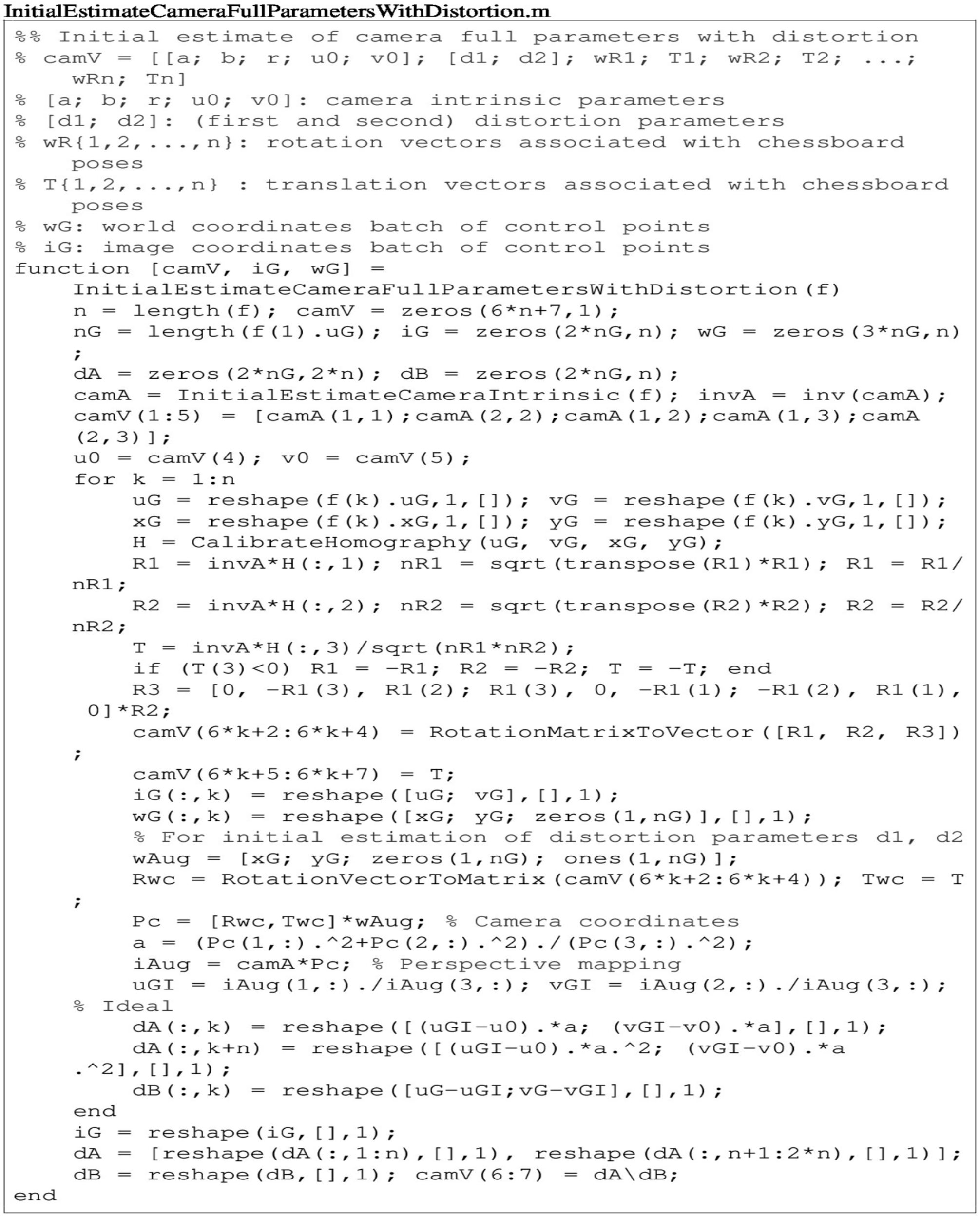
CameraIntrinsicCalibration.m

```
%% Read chessboard images and extract control points
CF = dir([imgDir ’*.jpg’]); f = []; nG = 12; wG = 0.02;
for k=1:length(CF)
    fk.img = rgb2gray(imread([CF(k).folder ’\’ CF(k).name])); f
    = [f, fk];
end
f = ExtractChessboardControlPoints(f, nG, wG);

%% Initial estimate of camera full parameters
[camV, iG, wG] = InitialEstimateCameraFullParameters(f);

%% Refinement of camera full parameters via optimization
camV = SolveParametricNonlinearEquation(...
            @CameraProjectionBatch, camV, wG, iG);
```

Matlab code for distortion-considered batch projection of the world coordinates (of control points) to the image coordinates is given as follows.



InitialEstimateCameraFullParametersWithDistortion.m

```
%% Initial estimate of camera full parameters with distortion
% camV = [[a; b; r; u0; v0]; [d1; d2]; wR1; T1; wR2; T2; ...;
    wRn; Tn]
% [a; b; r; u0; v0]: camera intrinsic parameters
% [d1; d2]: (first and second) distortion parameters
% wR{1,2,...,n}: rotation vectors associated with chessboard
    poses
% T{1,2,...,n} : translation vectors associated with chessboard
    poses
% wG: world coordinates batch of control points
% iG: image coordinates batch of control points
function [camV, iG, wG] =
    InitialEstimateCameraFullParametersWithDistortion(f)
    n = length(f); camV = zeros(6*n+7,1);
    nG = length(f(1).uG); iG = zeros(2*nG,n); wG = zeros(3*nG,n)
    ;
    dA = zeros(2*nG,2*n); dB = zeros(2*nG,n);
    camA = InitialEstimateCameraIntrinsic(f); invA = inv(camA);
    camV(1:5) = [camA(1,1);camA(2,2);camA(1,2);camA(1,3);camA
    (2,3)];
    u0 = camV(4); v0 = camV(5);
    for k = 1:n
        uG = reshape(f(k).uG,1,[]); vG = reshape(f(k).vG,1,[]);
        xG = reshape(f(k).xG,1,[]); yG = reshape(f(k).yG,1,[]);
        H = CalibrateHomography(uG, vG, xG, yG);
        R1 = invA*H(:,1); nR1 = sqrt(transpose(R1)*R1); R1 = R1/
    nR1;
        R2 = invA*H(:,2); nR2 = sqrt(transpose(R2)*R2); R2 = R2/
    nR2;
        T = invA*H(:,3)/sqrt(nR1*nR2);
        if (T(3)<0) R1 = -R1; R2 = -R2; T = -T; end
        R3 = [0, -R1(3), R1(2); R1(3), 0, -R1(1); -R1(2), R1(1),
     0]*R2;
        camV(6*k+2:6*k+4) = RotationMatrixToVector([R1, R2, R3])
    ;
        camV(6*k+5:6*k+7) = T;
        iG(:,k) = reshape([uG; vG],[],1);
        wG(:,k) = reshape([xG; yG; zeros(1,nG)],[],1);
        % For initial estimation of distortion parameters d1, d2
        wAug = [xG; yG; zeros(1,nG); ones(1,nG)];
        Rwc = RotationVectorToMatrix(camV(6*k+2:6*k+4)); Twc = T
    ;
        Pc = [Rwc,Twc]*wAug; % Camera coordinates
        a = (Pc(1,:).^2+Pc(2,:).^2)./(Pc(3,:).^2);
        iAug = camA*Pc; % Perspective mapping
        uGI = iAug(1,:)./iAug(3,:); vGI = iAug(2,:)./iAug(3,:);
    % Ideal
        dA(:,k) = reshape([(uGI-u0).*a; (vGI-v0).*a],[],1);
        dA(:,k+n) = reshape([(uGI-u0).*a.^2; (vGI-v0).*a
    .^2],[],1);
        dB(:,k) = reshape([uG-uGI;vG-vGI],[],1);
    end
    iG = reshape(iG,[],1);
    dA = [reshape(dA(:,1:n),[],1), reshape(dA(:,n+1:2*n),[],1)];
    dB = reshape(dB,[],1); camV(6:7) = dA\dB;
end
```

Once the parametric nonlinear function **CameraProjectionBatchWithDistortion.m** is defined, one can still resort to the Matlab code **SolveParametricNonlinearEquation.m** to refine camera intrinsic parameters with distortion parameters. Complete Matlab code for camera intrinsic calibration is given as follows.

CameraProjectionBatchWithDistortion.m

```
%% Camera Projection Batch with Distortion
% camV = [[a; b; r; u0; v0]; [d1; d2]; wR1; T1; wR2; T2; ...;
    wRn; Tn]
% [a; b; r; u0; v0]: camera intrinsic parameters
% [d1; d2]: (first and second) distortion parameters
% wR{1,2,...,n}: rotation vectors batch
% T{1,2,...,n} : translation vectors batch
% wG: given world coordinates batch
% iG: projected image coordinates batch
function iG = CameraProjectionBatchWithDistortion(camV, wG)
    n = size(wG,2); nG = size(wG,1)/3; iG = zeros(2*nG,n);
    camA = [camV(1), camV(3), camV(4); 0, camV(2), camV(5); 0,
    0, 1];
    d1 = camV(6); d2 = camV(7); u0 = camV(4); v0 = camV(5);
    for k = 1:n
        wAug = [wG(1:3:end,k)’;wG(2:3:end,k)’;wG(3:3:end,k)’;
    ones(1,nG)];
        Rwc = RotationVectorToMatrix(camV(6*k+2:6*k+4)); %
    Rotation matrix
        Twc = camV(6*k+5:6*k+7); % Translation vector
        Pc = [Rwc,Twc]*wAug; % Camera coordinates
        a = (Pc(1,:).^2+Pc(2,:).^2)./(Pc(3,:).^2);
        sa = 1+d1*a+d2*a.^2; % Distortion scaling factor
        iAug = camA*Pc; % Perspective mapping
        uGI = iAug(1,:)./iAug(3,:); vGI = iAug(2,:)./iAug(3,:);
    % Ideal
        uG = u0*ones(1,nG)+(uGI-u0).*sa; vG = v0*ones(1,nG)+(vGI
    -v0).*sa;
        iG(:,k) = reshape([uG;vG],[],1);
    end
    iG = reshape(iG,[],1);
end
```

The world coordinates of the control points in the multiple chessboard poses are projected onto corresponding images according to the calibrated camera parameters, as illustrated in Fig. 5.4, where image projections of the control points are marked by yellow circles. The image projections of the control points are intuitively consistent with their image coordinates ground-truth (i.e., their image coordinates extracted directly via image processing), which reflects the effectiveness of camera calibration.

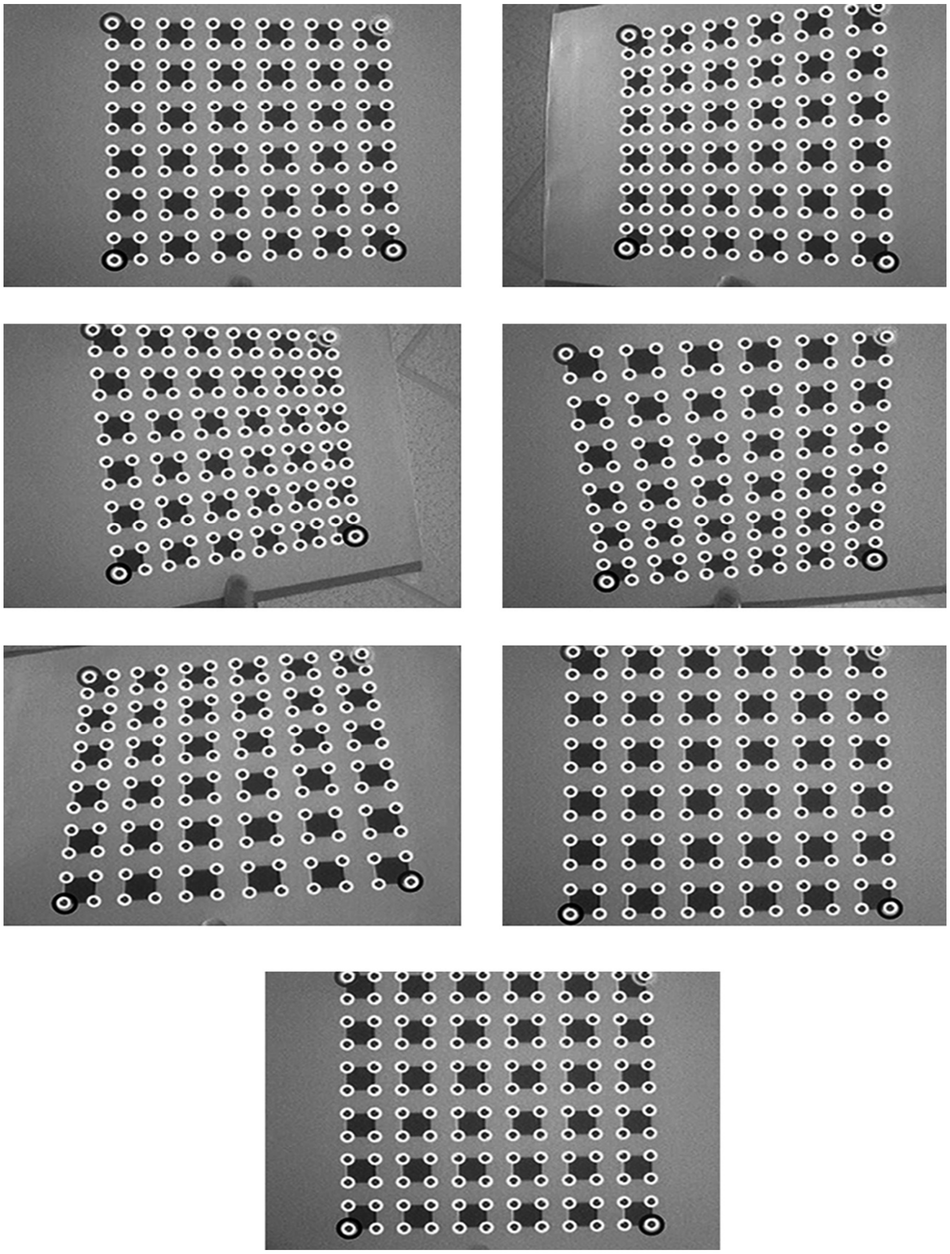


Fig. 5.4 Control points projected according to calibrated camera parameters

For quantitative evaluation of the effects of camera parameters refinement via optimization and of incorporating distortion parameters into the camera model, camera intrinsic parameters and projection residuals associated with various calibration configurations are listed in Tables 5.1 and 5.2.

Table 5.1 Calibrated camera intrinsic parameters and projection residuals

	α	β	γ	u_0	v_0	d_1	d_2	**proj. res.**
init. est. (n.)	849.0	848.9	−1.270	340.0	207.7	–	–	3731.4
refi. est. (n.)	850.9	849.6	0.061	340.1	207.7	–	–	3147.4
init. est. (d.)	849.0	848.9	−1.270	340.0	207.7	0.059	−0.928	3554.5
refi. est. (d.)	851.0	849.7	−0.223	340.2	207.6	−0.140	0.203	2874.4

Table 5.2 Camera calibration comparison via cross validation

	α	β	γ	u_0	v_0	d_1	d_2	**proj. res.**
refi. est. (n.)	850.5	849.7	−0.405	339.8	208.1	–	–	498.0
refi. est. (d.)	850.1	849.4	−0.713	340.0	207.9	−0.141	0.211	449.1

Concerning Table 5.1, control points of all the chessboard poses illustrated in Fig. 5.1 are used for camera calibration. The first and second rows of Table 5.1 give calibration results without considering distortion parameters ("n." means "non-distortion considered"), whereas the third and fourth rows give calibration results with distortion parameters considered ("d." means "distortion considered"). The first and third rows of Table 5.1 give the initial estimates of camera intrinsic parameters without and with distortion parameters considered respectively ("init. est." means "initial estimate"), whereas the second and fourth rows give the refined estimates of camera intrinsic parameters without and with distortion parameters considered respectively ("refi. est." means "refined estimate"). The last column of Table 5.1 gives the overall projection residuals corresponding to the four calibration configurations respectively ("proj. res." means "overall projection residual").

Table 5.1 conveys that without or with distortion parameters considered, the refined estimates of camera intrinsic parameters outperform (i.e., do "refine") the initial estimates of camera intrinsic parameters in terms of reducing the overall projection residual. Besides, the calibrated camera intrinsic parameters with distortion parameters considered outperform the calibrated camera intrinsic parameters without distortion parameters considered, also in terms of reducing the overall projection residual.

Note that a model with more parameters tends to "outperform" its counterpart with less parameters on the same data that are used both for model parameter estimation and for model parameter evaluation. To avoid demonstration of "fake advantage" of a model with more parameters over its counterpart with less parameters due to potential *over-fitting*, one can resort to *cross validation* for fair comparison.

More specifically, without and with distortion parameters considered, control points of the chessboard poses except the last one are used for camera intrinsic calibration, whereas control points of the last chessboard pose are used for overall projection residual based evaluation of the calibrated camera intrinsic parameters. Calibration results are given in Table 5.2, whereas the abbreviations have the same meanings as those in Table 5.1. As given in Table 5.2, the calibrated camera intrinsic parameters with distortion parameters considered still outperform the calibrated camera intrinsic parameters without distortion parameters considered, even when the experimental methodology of cross validation is adopted.

On one hand, the cross validation based comparative study demonstrated inTable 5.2 implies that the images illustrated in Fig. 5.1 do suffer from slight image distortion; otherwise, the calibrated camera intrinsic parameters with distortion parameters considered would not outperform the calibrated camera intrinsic parameters without distortion parameters considered, because the former took into account useless model factors.

On the other hand, one had better neither exaggerate the influence of slight image distortion nor exaggerate the advantage of the calibrated camera intrinsic parameters with distortion parameters considered over those without distortion parameters considered. After all, as demonstrated Tables 5.1 and 5.2, the calibration results without and with distortion parameters considered have only small differences mutually. Simply neglecting image distortion may be acceptable in some practical applications.

Even when image distortion is not negligible, as explained in Sect. 4.4 in Chap. 4, distortion rectification can be done to obtain rectified images exempt from image distortion, as if one could always handle ideal (distortion-free) images that conform to the pinhole camera model .

5.4 Camera-X Calibration

5.4.1 Fundamental Matrix Calibration

Given a pair of cameras that form stereo vision, to calibrate the relevant fundamental matrix, a preliminary step is to extract control points from both camera images and establish correspondences among control points. The SIFT and ORB features, for example, can assume the role of control points for fundamental matrix calibration. Denote the image coordinates of a set of control points in the first image coordinates system as

$$\boldsymbol{P}_{\mathrm{i},1}^{[1]} \equiv \begin{bmatrix} u_1^{[1]} \\ v_1^{[1]} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},2}^{[1]} \equiv \begin{bmatrix} u_2^{[1]} \\ v_2^{[1]} \end{bmatrix}, \quad \cdots \quad, \quad \boldsymbol{P}_{\mathrm{i},n}^{[1]} \equiv \begin{bmatrix} u_n^{[1]} \\ v_n^{[1]} \end{bmatrix},$$

and denote the image coordinates of their counterparts in the second image coordinates system as

$$\boldsymbol{P}_{\mathrm{i},1}^{[2]} \equiv \begin{bmatrix} u_1^{[2]} \\ v_1^{[2]} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},2}^{[2]} \equiv \begin{bmatrix} u_2^{[2]} \\ v_2^{[2]} \end{bmatrix}, \quad \cdots \quad, \quad \boldsymbol{P}_{\mathrm{i},n}^{[2]} \equiv \begin{bmatrix} u_n^{[2]} \\ v_n^{[2]} \end{bmatrix}.$$

Each pair of control points satisfy the epipolar constraint (4.62)

$$\begin{bmatrix} u_k^{[1]} & v_k^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} u_k^{[2]} \\ v_k^{[2]} \\ 1 \end{bmatrix} = 0$$

from which a homogeneous linear equation in terms of elements of the fundamental matrix (4.63) can be established as

$$\begin{aligned} \operatorname{vec}\left(\begin{bmatrix} u_k^{[1]} & v_k^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} u_k^{[2]} \\ v_k^{[2]} \\ 1 \end{bmatrix} \right) &= \quad 0 \\ \Longleftrightarrow \left(\begin{bmatrix} u_k^{[2]} & v_k^{[2]} & 1 \end{bmatrix} \otimes \begin{bmatrix} u_k^{[1]} & v_k^{[1]} & 1 \end{bmatrix} \right) \operatorname{vec}(\boldsymbol{F}) &= \quad 0. \end{aligned}$$

$$\begin{bmatrix} \begin{bmatrix} u_1^{[2]} & v_1^{[2]} & 1 \end{bmatrix} \otimes \begin{bmatrix} u_1^{[1]} & v_1^{[1]} & 1 \end{bmatrix} \\ \text{[6}pt\text{]}\begin{bmatrix} u_2^{[2]} & v_2^{[2]} & 1 \end{bmatrix} \otimes \begin{bmatrix} u_2^{[1]} & v_2^{[1]} & 1 \end{bmatrix} \\ \cdots \\ \begin{bmatrix} u_n^{[2]} & v_n^{[2]} & 1 \end{bmatrix} \otimes \begin{bmatrix} u_n^{[1]} & v_n^{[1]} & 1 \end{bmatrix} \end{bmatrix} \mathrm{vec}(\boldsymbol{F}) = \quad \mathbf{0} \tag{5.45}$$

where $\otimes$ denotes the *Kronecker product* which has already been described in Sect. 2.1 in Chap. 2.

Solve (5.45) via the method presented in Sect. F.2 in Appendix F. The solution of (5.45), let it be $\bar{\boldsymbol{F}}$, is normally of full rank in practical applications and hence violates the rank constraint that the fundamental matrix must have a rank of two (see Sect. 4.5.2 in Chap. 4). Force the rank constraint on the final solution of the fundamental matrix on one hand and make the final solution as close as possible to $\bar{\boldsymbol{F}}$ on the other hand, namely to solve the constrained optimization problem

$$\boldsymbol{F} = \mathrm{argmin}_{\boldsymbol{F}} \left\|\boldsymbol{F} - \bar{\boldsymbol{F}}\right\|_F^2 \qquad \text{s.t. } \mathrm{rank}(\boldsymbol{F}) = 2 \tag{5.46}$$

where the *Frobenius norm* is chosen as the matrix norm—The Frobenius norm may be regarded as the "L_2-norm" of the matrix.

The solution of (5.46) is obtained as follows: perform *singular value decomposition* of $\bar{\boldsymbol{F}}$ as $\bar{\boldsymbol{F}} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \sigma_2, \sigma_3)\boldsymbol{V}^{\mathrm{T}}$ or in details as

$$\bar{\boldsymbol{F}} = [\boldsymbol{U}_1 \quad \boldsymbol{U}_2 \quad \boldsymbol{U}_3] \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^{\mathrm{T}} \\ \boldsymbol{V}_2^{\mathrm{T}} \\ \boldsymbol{V}_3^{\mathrm{T}} \end{bmatrix}$$

where $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3 > 0$ (note that $\bar{\boldsymbol{F}}$ is normally of full rank), the 3-by-3 matrices $\boldsymbol{U} \equiv \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 & \boldsymbol{U}_3 \end{bmatrix}$ and $\boldsymbol{V} \equiv \begin{bmatrix} \boldsymbol{V}_1 & \boldsymbol{V}_2 & \boldsymbol{V}_3 \end{bmatrix}$ are orthonormal matrices. The solution of the fundamental matrix $\boldsymbol{F}$ is

$$\boldsymbol{F} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \sigma_2, 0)\boldsymbol{V}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^{\mathrm{T}} \\ \boldsymbol{V}_2^{\mathrm{T}} \end{bmatrix} = \sigma_1 \boldsymbol{U}_1 \boldsymbol{V}_1^{\mathrm{T}} + \sigma_2 \boldsymbol{U}_2 \boldsymbol{V}_2^{\mathrm{T}}. \tag{5.47}$$

Proof Consider a generalization of the constrained optimization problem (5.46)

$$\boldsymbol{F} = \mathrm{argmin}_{\boldsymbol{F}} \, \|\boldsymbol{F} - \bar{\boldsymbol{F}}\|_F^2 \qquad \mathrm{s.\,t.}\ \ \mathrm{rank}(\boldsymbol{F}) = m < n \tag{5.48}$$

where $\bar{\boldsymbol{F}} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \ldots, \sigma_n)\boldsymbol{V}^{\mathrm{T}}$ with $\sigma_1 \geqslant \cdots \geqslant \sigma_n > 0$. If the solution of (5.48) is proved to be

$$\boldsymbol{F} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \ldots, \sigma_m, 0, \ldots, 0)\boldsymbol{V}^{\mathrm{T}} = \begin{bmatrix}\boldsymbol{U}_1 & \ldots & \boldsymbol{U}_m\end{bmatrix} \begin{bmatrix}\sigma_1 & & \\ & \ddots & \\ & & \sigma_m\end{bmatrix} \begin{bmatrix}\boldsymbol{V}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{V}_m^{\mathrm{T}}\end{bmatrix}, \tag{5.49}$$

then the conclusion (5.47) holds naturally.

Denote $\boldsymbol{\Sigma}_\sigma \equiv \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$, $\boldsymbol{E} \equiv \boldsymbol{U}^{\mathrm{T}}\boldsymbol{F}\boldsymbol{V}$, and obtain

$$\begin{aligned}
\|\boldsymbol{F} - \bar{\boldsymbol{F}}\|_F^2 = \ & \mathrm{tr}\{(\boldsymbol{F} - \bar{\boldsymbol{F}})^{\mathrm{T}}(\boldsymbol{F} - \bar{\boldsymbol{F}})\} = \mathrm{tr}\{(\boldsymbol{F} - \boldsymbol{U}\boldsymbol{\Sigma}_\sigma\boldsymbol{V}^{\mathrm{T}})^{\mathrm{T}}(\boldsymbol{F} - \boldsymbol{U}\boldsymbol{\Sigma}_\sigma\boldsymbol{V}^{\mathrm{T}})\} \\
= \ & \mathrm{tr}\{[\boldsymbol{U}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{F}\boldsymbol{V} - \boldsymbol{\Sigma}_\sigma)\boldsymbol{V}^{\mathrm{T}}]^{\mathrm{T}}[\boldsymbol{U}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{F}\boldsymbol{V} - \boldsymbol{\Sigma}_\sigma)\boldsymbol{V}^{\mathrm{T}}]\} \\
= \ & \mathrm{tr}\{\boldsymbol{V}(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)^{\mathrm{T}}(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)\boldsymbol{V}^{\mathrm{T}}\} \\
= \ & \mathrm{tr}\{\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)^{\mathrm{T}}(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)\} \\
= \ & \mathrm{tr}\{(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)^{\mathrm{T}}(\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma)\} = \|\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma\|_F^2.
\end{aligned}$$

The *cyclic property* of trace operation [7] is used in above derivation, from which one can know that the constrained optimization problem (5.48) having the solution (5.49) is equivalent to the constrained optimization problem

$$\boldsymbol{E} = \mathrm{argmin}_{\boldsymbol{E}} \, \|\boldsymbol{E} - \boldsymbol{\Sigma}_\sigma\|_F^2 \qquad \mathrm{s.\,t.}\ \ \mathrm{rank}(\boldsymbol{E}) = m \tag{5.50}$$

having the solution

$$\boldsymbol{E} = \mathrm{diag}(\sigma_1, \ldots, \sigma_m, 0, \ldots, 0). \tag{5.51}$$

A generic matrix $\boldsymbol{E}$ of rank m can be expressed as

$$\boldsymbol{E} = \boldsymbol{u}\mathrm{diag}(\lambda_1, \ldots, \lambda_m, 0, \ldots, 0)\boldsymbol{v}^{\mathrm{T}} = \begin{bmatrix}\boldsymbol{u}_1 & \cdots & \boldsymbol{u}_m\end{bmatrix} \begin{bmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_m\end{bmatrix} \begin{bmatrix}\boldsymbol{v}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{v}_m^{\mathrm{T}}\end{bmatrix}$$

Denote $\boldsymbol{\Sigma}_\lambda \equiv \mathrm{diag}(\lambda_1, \ldots, \lambda_m, 0, \ldots, 0)$ and obtain

$$\begin{aligned}
\|\boldsymbol{E}-\boldsymbol{\Sigma}_\sigma\|_F^2 =&\quad \mathrm{tr}\{(\boldsymbol{E}-\boldsymbol{\Sigma}_\sigma)^{\mathrm{T}}(\boldsymbol{E}-\boldsymbol{\Sigma}_\sigma)\} = \mathrm{tr}\{\boldsymbol{E}^{\mathrm{T}}\boldsymbol{E}+\boldsymbol{\Sigma}_\sigma^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\} - 2\mathrm{tr}\{\boldsymbol{\Sigma}_\sigma\boldsymbol{E}\}\\
=&\quad \mathrm{tr}\{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\Sigma}_\lambda^2\boldsymbol{v}\} + \mathrm{tr}\{\boldsymbol{\Sigma}_\sigma^2\} - 2\mathrm{tr}\{\boldsymbol{\Sigma}_\sigma\boldsymbol{u}\boldsymbol{\Sigma}_\lambda\boldsymbol{v}^{\mathrm{T}}\}\\
=&\quad \mathrm{tr}\{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\Sigma}_\lambda^2\} + \mathrm{tr}\{\boldsymbol{\Sigma}_\sigma^2\} - 2\mathrm{tr}\left\{\boldsymbol{\Sigma}_\sigma \sum_{i=1}^m \lambda_i\boldsymbol{u}_i\boldsymbol{v}_i^{\mathrm{T}}\right\}\\
=&\quad \mathrm{tr}\{\boldsymbol{\Sigma}_\lambda^2\} + \mathrm{tr}\{\boldsymbol{\Sigma}_\sigma^2\} - 2\sum_{i=1}^m \lambda_i \mathrm{tr}\{\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\}\\
=&\quad \sum_{i=1}^m \lambda_i^2 + \sum_{j=1}^n \sigma_j^2 - 2\sum_{i=1}^m \lambda_i\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\\
=&\quad \sum_{i=1}^m \left(\lambda_i - \boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\right)^2 + \sum_{j=1}^n \sigma_j^2 - \sum_{i=1}^m \left(\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\right)^2\\
\geqslant&\quad \sum_{j=1}^n \sigma_j^2 - \sum_{i=1}^m \left(\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\right)^2.
\end{aligned}$$

According to the *Cauchy inequality*, each $\left(\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\right)^2$ satisfies

$$\left(\boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma\boldsymbol{u}_i\right)^2 = \left[(\boldsymbol{\Sigma}_\sigma\boldsymbol{v}_i)^{\mathrm{T}}\boldsymbol{u}_i\right]^2 \leqslant \|\boldsymbol{\Sigma}_\sigma\boldsymbol{v}_i\|_2^2 \cdot \|\boldsymbol{u}_i\|_2^2 = \|\boldsymbol{\Sigma}_\sigma\boldsymbol{v}_i\|_2^2 = \boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma^2\boldsymbol{v}_i,$$

so

$$\|\boldsymbol{E}-\boldsymbol{\Sigma}_\sigma\|_F^2 \quad \geqslant \sum_{j=1}^n \sigma_j^2 - \sum_{i=1}^m \boldsymbol{v}_i^{\mathrm{T}}\boldsymbol{\Sigma}_\sigma^2\boldsymbol{v}_i = \sum_{j=1}^n \sigma_j^2 - \sum_{i=1}^m \sum_{j=1}^n v_{ji}^2\sigma_j^2 = \sum_{j=1}^n \sigma_j^2 k_j,$$

where $k_j \equiv 1 - \sum_{i=1}^m v_{ji}^2 (j \in \{1,\ldots,n\})$ satisfy

$$\begin{aligned}
&0 \leqslant k_1 \leqslant 1, \quad 0 \leqslant k_2 \leqslant 1, \quad \cdots \quad , \quad 0 \leqslant k_n \leqslant 1,\\
&k_1 + k_2 + \cdots + k_n = \sum_{j=1}^n \left(1 - \sum_{i=1}^m v_{ji}^2\right) = n - \sum_{i=1}^m \sum_{j=1}^n v_{ji}^2 = n - m.
\end{aligned}$$

Then one has

$$\begin{aligned}
\|\boldsymbol{E}-\boldsymbol{\Sigma}_\sigma\|_F^2 \geqslant&\quad \sum_{j=1}^n \sigma_j^2 k_j = \sum_{j=m+1}^n \sigma_j^2 + \sum_{j=1}^m k_j\sigma_j^2 - \sum_{j=m+1}^n (1-k_j)\sigma_j^2\\
\geqslant&\quad \sum_{j=m+1}^n \sigma_j^2 + \sum_{j=1}^m k_j\sigma_m^2 - \sum_{j=m+1}^n (1-k_j)\sigma_m^2 = \sum_{j=m+1}^n \sigma_j^2\\
&\quad \left(\text{note } \sum_{j=1}^m k_j - \sum_{j=m+1}^n (1-k_j) = \sum_{j=1}^n k_j - (n-m) = 0\right)
\end{aligned}$$

where the equality holds when $\boldsymbol{E} = \mathrm{diag}(\sigma_1,\ldots,\sigma_m,0,\ldots,0)$. In other words, the constrained optimization problem (5.50) has the solution (5.51). The proof is done. □

Matlab code for fundamental matrix calibration is given as follows.

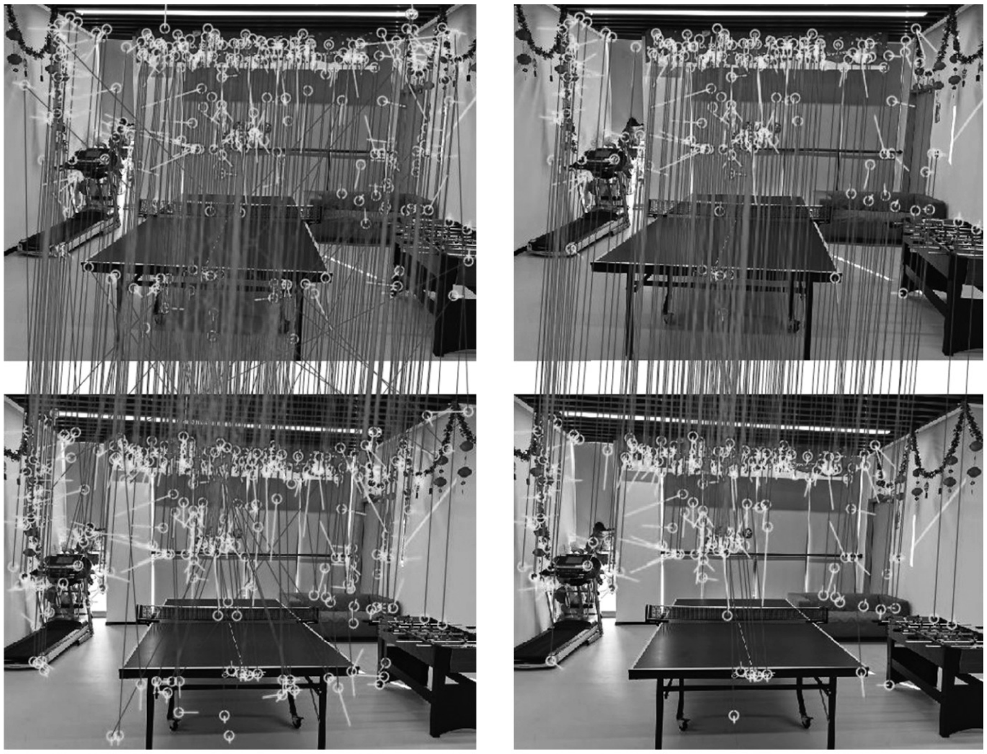
CameraIntrinsicCalibrationWithDistortion.m

```
%% Read chessboard images and extract control points
CF = dir([imgDir ’*.jpg’]); f = []; nG = 12; wG = 0.02;
for k=1:length(CF)
    fk.img = rgb2gray(imread([CF(k).folder ’\’ CF(k).name])); f
    = [f, fk];
end
f = ExtractChessboardControlPoints(f, nG, wG);

%% Initial estimate of camera full parameters with distortion
 [camV, iG, wG] =
    InitialEstimateCameraFullParametersWithDistortion(f);

%% Refinement of camera full parameters with distortion
camV = SolveParametricNonlinearEquation(...
                @CameraProjectionBatchWithDistortion, camV, wG,
    iG);
```

As mentioned previously, a variety of features such as SIFT features and ORB features can assume the role of control points for fundamental matrix calibration. For example, adopt SIFT features as control points and consider the same pair of camera images illustrated in Fig. 4.17. Extract SIFT features from both the left and right images and follow the SIFT descriptor similarity criterion to establish raw correspondences of SIFT features between the two images, as illustrated in Fig. 5.5a. Then apply the RANSAC (RANdom SAmple Consensus) method to obtain refined correspondences of SIFT features between the two images, as illustrated in Fig. 5.5b—Presentation of the RANSAC method is postponed to Chap. 6.



(a) (b)

Fig. 5.5 Control points for fundamental matrix calibration **a** raw correspondences of SIFT features; **b** RANSAC refined correspondences of SIFT features

Once the refined correspondences of SIFT features are obtained, they can be used as control points to calibrate the fundamental matrix via (5.45) and (5.47). Once the fundamental matrix $\boldsymbol{F}$ is calibrated, epipolar lines passing arbitrarily selected points can be computed accordingly. The effect of several example pairs of epipolar lines computed according to the calibrated fundamental matrix is demonstrated in Fig. 4.17.

5.4.2 Essential Matrix Calibration

The equation (4.66)

$$\begin{aligned} \boldsymbol{E} = \boldsymbol{A}^{[1]\mathrm{T}}\boldsymbol{F}\boldsymbol{A}^{[2]} &= \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix} \\ \Leftrightarrow \quad \boldsymbol{F} = \left(\boldsymbol{A}^{[1]}\right)^{-\mathrm{T}}\boldsymbol{E}\left(\boldsymbol{A}^{[2]}\right)^{-1} & \\ = \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix}^{-1} & \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \end{aligned}$$

conveys that the essential matrix $\boldsymbol{E}$ and the fundamental matrix $\boldsymbol{F}$ can be mutually transformed in bijective way. So calibrating the essential matrix is theoretically equivalent to calibrating the fundamental matrix, yet the former tends to be numerically more stable in practice.

Suppose the two camera intrinsic matrices $\boldsymbol{A}^{[1]}$ and $\boldsymbol{A}^{[2]}$ are calibrated a priori and hence known. Still denote the image coordinates of a set of control points in the first image coordinates system as

$$\boldsymbol{P}_{\mathrm{i},1}^{[1]} \equiv \begin{bmatrix} u_1^{[1]} \\ v_1^{[1]} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},2}^{[1]} \equiv \begin{bmatrix} u_2^{[1]} \\ v_2^{[1]} \end{bmatrix}, \quad \cdots \quad, \quad \boldsymbol{P}_{\mathrm{i},n}^{[1]} \equiv \begin{bmatrix} u_n^{[1]} \\ v_n^{[1]} \end{bmatrix},$$

and denote the image coordinates of their counterparts in the second image coordinates system as

$$\boldsymbol{P}_{\mathrm{i},1}^{[2]} \equiv \begin{bmatrix} u_1^{[2]} \\ v_1^{[2]} \end{bmatrix}, \quad \boldsymbol{P}_{\mathrm{i},2}^{[2]} \equiv \begin{bmatrix} u_2^{[2]} \\ v_2^{[2]} \end{bmatrix}, \quad \cdots \quad, \quad \boldsymbol{P}_{\mathrm{i},n}^{[2]} \equiv \begin{bmatrix} u_n^{[2]} \\ v_n^{[2]} \end{bmatrix}.$$

Compute their z_{c}-normalized camera coordinates

$$\begin{bmatrix} x_{\mathrm{c},k}^{[1]} \\ y_{\mathrm{c},k}^{[1]} \\ z_{\mathrm{c},k}^{[1]} \end{bmatrix} \equiv \boldsymbol{P}_{\mathrm{c},k}^{[1]} = \left(\boldsymbol{A}^{[1]}\right)^{-1} \boldsymbol{P}_{\mathrm{i},k}^{[1]A} = \begin{bmatrix} \alpha^{[1]} & \gamma^{[1]} & u_0^{[1]} \\ 0 & \beta^{[1]} & v_0^{[1]} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u_k^{[1]} \\ v_k^{[1]} \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} x_{\mathrm{c},k}^{[2]} \\ y_{\mathrm{c},k}^{[2]} \\ z_{\mathrm{c},k}^{[2]} \end{bmatrix} \equiv \boldsymbol{P}_{\mathrm{c},k}^{[2]} = \left(\boldsymbol{A}^{[2]}\right)^{-1} \boldsymbol{P}_{\mathrm{i},k}^{[2]A} = \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u_k^{[2]} \\ v_k^{[2]} \\ 1 \end{bmatrix},$$

where $k \in \{1, \ldots, n\}$.

Each pair of control points satisfy the epipolar constraint (4.65)

$$\begin{bmatrix} x_{\mathrm{c},k}^{[1]} & y_{\mathrm{c},k}^{[1]} & z_{\mathrm{c},k}^{[1]} \end{bmatrix} \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} x_{\mathrm{c},k}^{[2]} \\ y_{\mathrm{c},k}^{[2]} \\ z_{\mathrm{c},k}^{[2]} \end{bmatrix} = 0$$

from which a homogeneous linear equation in terms of elements of the essential matrix (4.66) can be established as

$$\begin{aligned}\mathrm{vec}\left(\begin{bmatrix} x_{\mathrm{c},k}^{[1]} & y_{\mathrm{c},k}^{[1]} & z_{\mathrm{c},k}^{[1]}\end{bmatrix}\begin{bmatrix}E_{11}&E_{12}&E_{13}\\E_{21}&E_{22}&E_{23}\\E_{31}&E_{32}&E_{33}\end{bmatrix}\begin{bmatrix}x^{[2]}_{\mathrm{c},k}\\y^{[2]}_{\mathrm{c},k}\\z^{[2]}_{\mathrm{c},k}\end{bmatrix}\right) &= \quad 0\\ \iff \left(\begin{bmatrix} x_{\mathrm{c},k}^{[2]} & y_{\mathrm{c},k}^{[2]} & z_{\mathrm{c},k}^{[2]}\end{bmatrix}\otimes\begin{bmatrix} x_{\mathrm{c},k}^{[1]} & y_{\mathrm{c},k}^{[1]} & z_{\mathrm{c},k}^{[1]}\end{bmatrix}\right)\mathrm{vec}(\boldsymbol{E}) &= \quad 0.\end{aligned}$$

$$\begin{bmatrix}\begin{bmatrix} x_{\mathrm{c},1}^{[2]} & y_{\mathrm{c},1}^{[2]} & z_{\mathrm{c},1}^{[2]}\end{bmatrix}\otimes\begin{bmatrix} x_{\mathrm{c},1}^{[1]} & y_{\mathrm{c},1}^{[1]} & z_{\mathrm{c},1}^{[1]}\end{bmatrix}\\ [6pt]\begin{bmatrix} x_{\mathrm{c},2}^{[2]} & y_{\mathrm{c},2}^{[2]} & z_{\mathrm{c},2}^{[2]}\end{bmatrix}\otimes\begin{bmatrix} x_{\mathrm{c},2}^{[1]} & y_{\mathrm{c},2}^{[1]} & z_{\mathrm{c},2}^{[1]}\end{bmatrix}\\ \cdots \\ \begin{bmatrix} x_{\mathrm{c},n}^{[2]} & y_{\mathrm{c},n}^{[2]} & z_{\mathrm{c},n}^{[2]}\end{bmatrix}\otimes\begin{bmatrix} x_{\mathrm{c},n}^{[1]} & y_{\mathrm{c},n}^{[1]} & z_{\mathrm{c},n}^{[1]}\end{bmatrix}\end{bmatrix}\mathrm{vec}(\boldsymbol{E}) = \quad \mathbf{0}. \tag{5.52}$$

The way to establish (5.52) is similar to the way to establish (5.45).

Like how to solve the fundamental matrix, solve the essential matrix as follows: Solve (5.52) via the method presented in Sect. F.2 in Appendix F. Let the solution of (5.52) be $\bar{\boldsymbol{E}}$. Perform singular value decomposition of $\bar{\boldsymbol{E}}$ as $\bar{\boldsymbol{E}} = \boldsymbol{U}\mathrm{diag}(\sigma_1, \sigma_2, \sigma_3)\boldsymbol{V}^{\mathrm{T}}$ or in details as

$$\bar{\boldsymbol{E}} = [\boldsymbol{U}_1 \quad \boldsymbol{U}_2 \quad \boldsymbol{U}_3]\begin{bmatrix}\sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_1^{\mathrm{T}}\\ \boldsymbol{V}_2^{\mathrm{T}}\\ \boldsymbol{V}_3^{\mathrm{T}}\end{bmatrix}$$

where $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3 > 0$ (note that $\bar{\boldsymbol{E}}$ is normally of full rank). Then according to the constraint described in (4.75), the essential matrix $\boldsymbol{E}$ can be obtained as

$$\begin{aligned}\boldsymbol{E} = &\quad \boldsymbol{U}\mathrm{diag}\left(\tfrac{\sigma_1+\sigma_2}{2}, \tfrac{\sigma_1+\sigma_2}{2}, 0\right)\boldsymbol{V}^{\mathrm{T}}\\ = &\quad [\boldsymbol{U}_1 \quad \boldsymbol{U}_2]\begin{bmatrix}\frac{\sigma_1+\sigma_2}{2} & \\ & \frac{\sigma_1+\sigma_2}{2}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_1^{\mathrm{T}}\\ [9pt]\boldsymbol{V}_2^{\mathrm{T}}\end{bmatrix}\\ = &\quad \tfrac{\sigma_1+\sigma_2}{2}(\boldsymbol{U}_1\boldsymbol{V}_1^{\mathrm{T}} + \boldsymbol{U}_2\boldsymbol{V}_2^{\mathrm{T}}).\end{aligned} \tag{5.53}$$

The author abuses notations in (5.53) and (5.47) for simplicity, yet it is worth noting that what notations in (5.53) and (5.47) refer to are different.

Matlab code for essential matrix calibration is given as follows.[7]

CalibrateFundamentalMatrix.m

```
%% Calibrate the fundamental matrix between a pair of cameras
% Fm: Fundamental matrix [uai;vai;1]'*Fm*[ubi;vbi;1] = 0
% ua : u-coordinates of feature points in the first image
% va : v-coordinates of feature points in the first image
% ub : u-coordinates of feature points in the second image
% vb : v-coordinates of feature points in the second image
function Fm = CalibrateFundamentalMatrix(ua, va, ub, vb)
    nCP = length(ua); L = zeros(nCP,9);
    for k = 1:nCP
        L(k,:) = kron([ub(k), vb(k), 1], [ua(k), va(k), 1]);
    end
    fr = SolveHomogeneousLinearEquation(L);
    Fm = reshape(fr,3,3);
    [U,S,V] = svd(Fm); S(3,3)=0; Fm = U*S*V';
end
```

Once the essential matrix $\boldsymbol{E}$ is calibrated, the fundamental matrix $\boldsymbol{F}$ can be recovered as well via (4.66) namely

$$\begin{aligned}\boldsymbol{F} = &\ \left(\boldsymbol{A}^{[1]}\right)^{-\mathrm{T}}\boldsymbol{E}\left(\boldsymbol{A}^{[2]}\right)^{-1}\\ = &\begin{bmatrix}\alpha^{[1]} & 0 & 0\\ \gamma^{[1]} & \beta^{[1]} & 0\\ u_0^{[1]} & v_0^{[1]} & 1\end{bmatrix}^{-1}\begin{bmatrix}E_{11} & E_{12} & E_{13}\\ E_{21} & E_{22} & E_{23}\\ E_{31} & E_{32} & E_{33}\end{bmatrix}\begin{bmatrix}\alpha^{[2]} & \gamma^{[2]} & u_0^{[2]}\\ 0 & \beta^{[2]} & v_0^{[2]}\\ 0 & 0 & 1\end{bmatrix}^{-1}.\end{aligned}$$

The author recommends this indirect way of fundamental matrix calibration instead of the direct way presented in Sect. 5.4.1, because the indirect way is more desirable in terms of numeric stability.[8]

5.4.3 Stereo Camera Calibration

One may naturally extend the practice of single camera calibration formalized in (5.39) to the practice of stereo camera calibration formalized as a group of nonlinear equations in terms of free variables $\{\boldsymbol{v}_{\boldsymbol{A}^{[1]}}, \boldsymbol{v}_{\boldsymbol{A}^{[2]}}, \boldsymbol{w}_{\mathrm{cc}}^{[1,2]}, \boldsymbol{T}_{\mathrm{cc}}^{[1,2]}, \boldsymbol{w}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$:

$$\frac{\begin{bmatrix}\boldsymbol{e}_1^{\mathrm{T}}\\ \boldsymbol{e}_2^{\mathrm{T}}\end{bmatrix}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}^{[1]}})\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]}\end{bmatrix}\boldsymbol{P}_{\mathrm{w},l}^{[k]A}}{\boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}^{[1]}})\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]}\end{bmatrix}\boldsymbol{P}_{\mathrm{w},l}^{[k]A}} = \boldsymbol{P}_{\mathrm{i},l}^{[k][1]}, \tag{5.54a}$$

$$\frac{\begin{bmatrix}\boldsymbol{e}_1^{\mathrm{T}}\\ \boldsymbol{e}_2^{\mathrm{T}}\end{bmatrix}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}^{[2]}})\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{cc}}^{[1,2]}) & \boldsymbol{T}_{\mathrm{cc}}^{[1,2]}\end{bmatrix}\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]}\\ 0 & 1\end{bmatrix}\boldsymbol{P}_{\mathrm{w},l}^{[k]A}}{\boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{A}(\boldsymbol{v}_{\boldsymbol{A}^{[2]}})\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{cc}}^{[1,2]}) & \boldsymbol{T}_{\mathrm{cc}}^{[1,2]}\end{bmatrix}\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]}\\ 0 & 1\end{bmatrix}\boldsymbol{P}_{\mathrm{w},l}^{[k]A}} = \boldsymbol{P}_{\mathrm{i},l}^{[k][2]}, \tag{5.54b}$$

where $k \in \{1, \ldots, n\}$ and $l \in \{1, \ldots, m\}$ have the same meanings as in (5.39). $\boldsymbol{v}_{\boldsymbol{A}^{[1]}}$ and $\boldsymbol{v}_{\boldsymbol{A}^{[2]}}$ denote the intrinsic parameters of the first and second cameras respectively. The variables $\{\boldsymbol{w}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$ denote rotation vectors and translation vectors of chessboard poses with respect to the first camera coordinates system. The rotation vector $\boldsymbol{w}_{\mathrm{cc}}^{[1,2]}$ and the translation vector $\boldsymbol{T}_{\mathrm{cc}}^{[1,2]}$ represent the fixed transformation from the first camera coordinates system to the second camera coordinates system. The superscript, "[1]" and "[2]" attached to $\boldsymbol{P}_{\mathrm{i},l}^{[k]}$ have the meaning of being in the first and second image coordinates systems respectively.

Compare (5.39) and (5.54): The first equation of (5.54) is identical to (5.39), if the superscript "[1]" is omitted. The second equation of (5.54), which seems to have extra terms $\left[\boldsymbol{R}(\boldsymbol{w}_{\mathrm{cc}}^{[1,2]}) \quad \boldsymbol{T}_{\mathrm{cc}}^{[1,2]}\right]$, is essentially the same to (5.39). This can be understood if one bears in mind that

$$\left[\boldsymbol{R}(\boldsymbol{w}_{\mathrm{cc}}^{[1,2]}) \quad \boldsymbol{T}_{\mathrm{cc}}^{[1,2]}\right]\begin{bmatrix}\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) & \boldsymbol{T}_{\mathrm{wc}}^{[k]} \\ 0 & 1\end{bmatrix}$$

actually denote $\left[\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}^{[k]}) \quad \boldsymbol{T}_{\mathrm{wc}}^{[k]}\right]$ with respect to the second camera coordinates system. If the variables $\{\boldsymbol{w}_{\mathrm{wc}}^{[1]}, \boldsymbol{T}_{\mathrm{wc}}^{[1]}, \ldots, \boldsymbol{w}_{\mathrm{wc}}^{[n]}, \boldsymbol{T}_{\mathrm{wc}}^{[n]}\}$ denote rotation vectors and translation vectors of chessboard poses with respect to the second camera coordinates system instead of the first camera coordinates system, then the second equation of (5.54) will become identical to (5.39). The initial solution of the group of nonlinearequations (5.54) can also be obtained using the method presented in Sect. 5.1.

The practice of stereo camera calibration formalized in (5.54) can be furtherextended to the practice of calibrating a system of multi-view ($\geqslant 3$ views) stereo vision [8].

5.4.4 Camera-Range Sensor Calibration

Cameras and range sensors such as laser scanners and lidars are two important kinds of perceptive sensors for mobile robots especially intelligent vehicles. Multi-sensor fusion architectures, i.e. perception systems based on fusion of both kinds of sensors [9–11] are usually much more effective and robust than those based only on cameras or range sensors. To take advantage of camera-range sensor fusion, one first needs to realize camera-range sensor calibration (or co-calibration) [12, 13]. This is indispensable to association of the two heterogeneous kinds of sensor data namely visual data and range data.

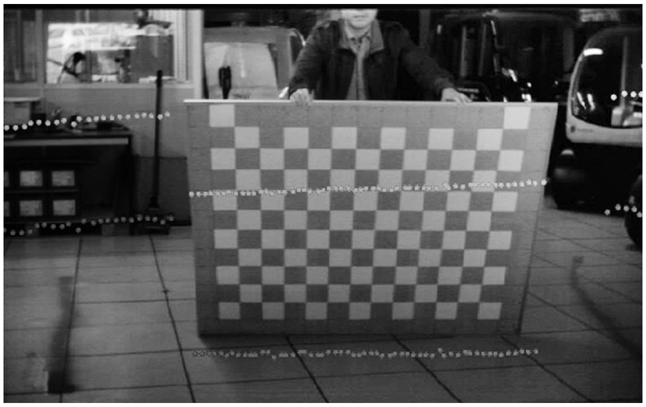


Fig. 5.6 Camera-laser scanner calibration [13]: laser points and their ground projections

An example of calibration effect of a camera and a two-dimensional laser scanner is illustrated in Fig. 5.6. Laser points are not physically visible to humans, yet thanks to the camera-laser scanner calibration, laser points can be correctly projected onto the image to form an augmented reality effect as if one could directly see the laser points.

References

1. Z. Zhang, A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1330–1334 (2000)
[ADS][Crossref]
2. O. Rodrigues, Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. J. de Mathématiques Pures et Appliquées **5**, 380–440 (1840)
3. S. Chern, W. Chen. *Lectures on Differential Geometry (in Chinese)* (Peking University Press, 1983)
4. S. Lang. *Algebra* (Springer, 2002)
5. K. Levenberg, A method for the solution of certain non-linear problems in least squares. Quarter. Appl. Math. **2**(2), 164–168 (1944)
[MathSciNet][Crossref][zbMATH]
6. D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters. J. Soc. Ind. Appl. Math. **11**(2), 431–441 (1963)
[MathSciNet][Crossref][zbMATH]
7. R.A. Horn, C.R. Johnson. *Matrix Analysis* (Cambridge University Press, 2012)
8. Y. Furukawa, J. Ponce, Accurate camera calibration from multi-view stereo and bundle adjustment. Int. J. Comput. Vis. **84**, 257–268 (2009)
[Crossref]
9. H. Li, M. Tsukada, F. Nashashibi, M. Parent, Multivehicle cooperative local mapping: a methodology based on occupancy grid map merging. IEEE Trans. Intell. Transport. Syst. **15**(5), 2089–2100 (2014)
[Crossref]
10. S. Pang, D. Morris, H. Radha. Clocs: camera-lidar object candidates fusion for 3d object detection, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2020), pp. 10386–10393
11. X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, C. Tai. Transfusion: robust lidar-camera fusion for 3d object detection with transformers, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 1090–1099
12. Q. Zhang, R. Pless, Extrinsic calibration of a camera and laser range finder (improves camera calibration), in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2004), pp. 2301–2306

13. H. Li, F. Nashashibi. *Comprehensive extrinsic calibration of a camera and a 2d laser scanner for a ground vehicle*. INRIA Tech Report, RT-438 (2013)

Footnotes

[1] In fact, the chessboard pattern does not need to be a strict chessboard, but can be any pattern of regularly arranged black squares for sake of control point extraction and registration convenience.

[2] See definition of the *skew-symmetric matrix* in Sect. D.2.1 in Appendix D.

[3] In the original article of *Rodrigues* [2], the *Rodrigues formula* of the rotation matrix is neither expressed explicitly nor formalized in the compact matrix form (5.20), but is implicitly reflected in a triplet of equations (in its original notations)

$$\begin{aligned} \Delta x = &\quad \sin\theta(y\cos l - z\cos h) + 2\sin^2(\theta/2)[\cos g(x\cos g + y\cos h + z\cos l) - x], \\ \Delta y = &\quad \sin\theta(z\cos g - x\cos l) + 2\sin^2(\theta/2)[\cos h(x\cos g + y\cos h + z\cos l) - y], \\ \Delta z = &\quad \sin\theta(x\cos h - y\cos g) + 2\sin^2(\theta/2)[\cos l(x\cos g + y\cos h + z\cos l) - z], \end{aligned}$$

which convey essentially the same formula of the rotation matrix. One may interpret original notations in above equations as follow: $(x+\Delta x, y+\Delta y, z+\Delta z)$ denotes the new position of the point (x, y, z) after rotation around the rotation axis $(-\cos g, -\cos h, -\cos l)$ by the angle θ.

[4] A Lie group is a continuous group that is a differential manifold of certain dimension as well [3].

[5] Derivation of (5.20) itself does not necessarily involve physical intuition of rotation laws, but only involves series expansions of the matrix exponential function, the sine function, and the cosine function.

[6] Lie algebra is a vector space with *multiplication* that satisfies

- *Distributive law*: $[a_1\boldsymbol{X}_1 + a_2\boldsymbol{X}_2, \boldsymbol{Y}] = a_1[\boldsymbol{X}_1, \boldsymbol{Y}] + a_2[\boldsymbol{X}_2, \boldsymbol{Y}]$;
- *Anticommutative law*: $[\boldsymbol{X}, \boldsymbol{Y}] = -[\boldsymbol{Y}, \boldsymbol{X}]$;
- *Jacobi identity*: $[\boldsymbol{X}, [\boldsymbol{Y}, \boldsymbol{Z}]] + [\boldsymbol{Y}, [\boldsymbol{Z}, \boldsymbol{X}]] + [\boldsymbol{Z}, [\boldsymbol{X}, \boldsymbol{Y}]] = 0$.

The formalism of the *Poisson bracket* $[,]$ is abused in above definition to denote a generic multiplication in Lie algebra, because the *Poisson bracket* $[\boldsymbol{X}, \boldsymbol{Y}] \equiv \boldsymbol{XY} - \boldsymbol{YX}$ is a representative and general binary operation satisfying above three conditions.

[7] In the demonstrated script **CalibrateEssentialMatrix.m**, the camera intrinsic matrices associated with the pair of images are assumed identical. Such assumption is fair for many practical applications. If the assumption cannot be adopted, then just add one more camera intrinsic matrix into the argument list of the function.

[8] If camera intrinsic matrices are unavailable and hence one is obliged to calibrate the fundamental matrix using the direct way, then the author recommends a preliminary step of normalizing the image coordinates before fundamental matrix calibration.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_6

6. Pattern Recognition for Feature Organization

Hao Li[1]
(1) Shanghai Jiao Tong University, Shanghai, China

6.1 Clustering
6.1.1 Direct Connection
6.1.2 K-Means
6.1.3 Mean Shift
6.1.4 Normalized Cut
6.1.5 Segmentation
6.2 Known Feature-Model Relationship Oriented Model Fitting
6.2.1 Least Mean Squares (LMS) Method
6.2.2 Recursive Linear LMS
6.2.3 Regularized LMS
6.3 Unknown Feature-Model Relationship Oriented Model Fitting
6.3.1 Hough Transform
6.3.2 Expectation Maximization (EM)
6.3.3 Random Sample Consensus (RANSAC)
References

In practical applications, sometimes extracted feature points are handled directly, whereas sometimes extracted feature points are organized into meaningful geometric structures namely **patterns**. This organizing process is called **pattern recognition**. As early as one century ago, researchers already pointed out the existence and importance of perceptual organization in biological vision systems [1]. By analogy, pattern recognition also plays an indispensable role in computer vision systems.

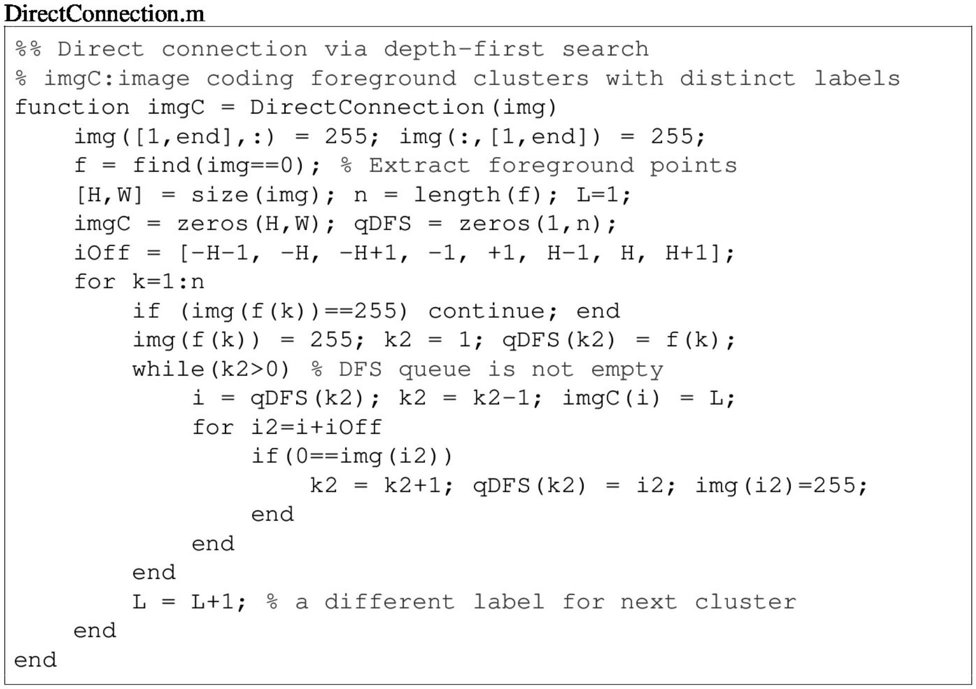
6.1 Clustering

A methodology of pattern recognition is to organize feature points into **clusters** according to their internal relationship.

6.1.1 Direct Connection

Probably the simplest way of clustering is to directly connect pixels of the same cluster via *depth-first search*. Direct connection has already been applied in the double thresholding step of *Canny edge detection* to cluster strong edge points and weak edge points into edge curve segments that at least contain strong edge points. In the context of Canny edge detection, it is unnecessary to distinguish among edge curve segments connected respectively via depth-first search.

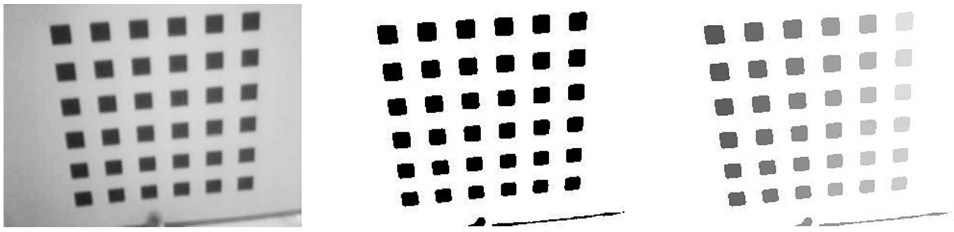
If one needs to distinguish among clusters of pixels. One may select an arbitrary foreground pixel, initiate depth-first search from it to connect whatever foreground pixels that connect to it, and give the connected cluster a label. Then select an arbitrary foreground pixel among remaining foreground pixels, initiate depth-first search from it to connect whatever foreground pixels that connect to it, and give this second connected cluster another label. Continue such process again and again on remaining foreground pixels to form clusters and give them distinct labels. Matlab code for direct connection via depth-first search is given as follows.



DirectConnection.m

```
%% Direct connection via depth-first search
% imgC:image coding foreground clusters with distinct labels
function imgC = DirectConnection(img)
    img([1,end],:) = 255; img(:,[1,end]) = 255;
    f = find(img==0); % Extract foreground points
    [H,W] = size(img); n = length(f); L=1;
    imgC = zeros(H,W); qDFS = zeros(1,n);
    iOff = [-H-1, -H, -H+1, -1, +1, H-1, H, H+1];
    for k=1:n
        if (img(f(k))==255) continue; end
        img(f(k)) = 255; k2 = 1; qDFS(k2) = f(k);
        while(k2>0) % DFS queue is not empty
            i = qDFS(k2); k2 = k2-1; imgC(i) = L;
            for i2=i+iOff
                if(0==img(i2))
                    k2 = k2+1; qDFS(k2) = i2; img(i2)=255;
                end
            end
        end
        L = L+1; % a different label for next cluster
    end
end
```

An example of direct connection effect is demonstrated in Fig. 6.1, where the subfigure(a) illustrates the original image smoothed by the Gaussian filter . Figure 6.1b illustrates an adaptively thresholded result of the smoothed original image (see Sect. 2.2.3 in Chap. 2 for presentation of adaptive thresholding), namely a binary image with the foreground indicated by black pixels. Figure 6.1c demonstrates the effect of direct connection via depth-first search, where the foreground clusters are distinguished from each other by gray-level differences.



(a) (b) (c)

Fig. 6.1 Direct connection via depth-first search **a** smoothed original image; **b** binary image with black foreground; **c** clusters distinguished by gray-levels

6.1.2 K-Means

The simple method of clustering namely the direct connection method presented previously is applicable to feature points whose mutual topological relationships are known explicitly or implicitly. In other words, if one knows a priori each feature point is connected to which other feature points, then one can apply the direct connection method to cluster the feature points. On the other hand, in most practical applications, there is no a priori knowledge of mutual topological relationships among feature points. So the direct connection method is far away from being generally applicable and is usually limited to handling of simple tasks. More sophisticated clustering methods are needed for practical applications.

The **K-means** method [2, 3] is a representative method for clustering feature points into a specified number of clusters. For a generic number K, call a way of clustering feature points into K clusters as K-*clustering*. Given a set of N feature points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_N$ intended for K-clustering, consider the optimization problem

$$\{\boldsymbol{K},\boldsymbol{m}\} = \underset{\boldsymbol{K},\boldsymbol{m}}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\boldsymbol{P}_j - \boldsymbol{m}_i\|_2^2, \tag{6.1}$$

where

$$\boldsymbol{K} \equiv \{\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_K\}$$

denotes a generic K-clustering of the N feature points and

$$\boldsymbol{m} \equiv \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_K\}$$

denotes K clustering points associated with the K clusters of $\boldsymbol{K}$ respectively. The solution of the optimization problem (6.1) means the optimal K-clustering by which the N feature points are most concentrated towards their respective clustering points on the whole.[1]

Denote the centroids of the K clusters of $\boldsymbol{K}$ respectively as $\bar{\boldsymbol{K}}_1$, $\bar{\boldsymbol{K}}_2$, ..., $\bar{\boldsymbol{K}}_K$, i.e.

$$\bar{\boldsymbol{K}}_1 \equiv \frac{1}{|\boldsymbol{K}_1|} \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_1} \boldsymbol{P}_j, \quad \bar{\boldsymbol{K}}_2 \equiv \frac{1}{|\boldsymbol{K}_2|} \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_2} \boldsymbol{P}_j, \quad \ldots \quad , \quad \bar{\boldsymbol{K}}_K \equiv \frac{1}{|\boldsymbol{K}_K|} \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_K} \boldsymbol{P}_j.$$

For each $i \in \{1, \ldots, K\}$ one has

$$\begin{aligned}
\sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\boldsymbol{P}_j - \boldsymbol{m}_i\|_2^2 &= \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} (\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i + \bar{\boldsymbol{K}}_i - \boldsymbol{m}_i)^{\mathrm{T}} (\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i + \bar{\boldsymbol{K}}_i - \boldsymbol{m}_i) \\
&= \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i\|_2^2 + \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\bar{\boldsymbol{K}}_i - \boldsymbol{m}_i\|_2^2 + 2(\bar{\boldsymbol{K}}_i - \boldsymbol{m}_i)^{\mathrm{T}} \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} (\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i) \\
&= \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i\|_2^2 + |\boldsymbol{K}_i| \cdot \|\bar{\boldsymbol{K}}_i - \boldsymbol{m}_i\|_2^2 \geqslant \sum_{\boldsymbol{P}_j \in \boldsymbol{K}_i} \|\boldsymbol{P}_j - \bar{\boldsymbol{K}}_i\|_2^2.
\end{aligned}$$

So the objective function specified in (6.1) satisfies the inequality

$$\sum_{i=1}^{K}\sum_{\boldsymbol{P}_j\in\boldsymbol{K}_i}\|\boldsymbol{P}_j-\boldsymbol{m}_i\|_2^2 \geqslant \sum_{i=1}^{K}\sum_{\boldsymbol{P}_j\in\boldsymbol{K}_i}\|\boldsymbol{P}_j-\bar{\boldsymbol{K}}_i\|_2^2. \tag{6.2}$$

The equality of (6.2) holds if and only if

$$\boldsymbol{m}_i=\bar{\boldsymbol{K}}_i \quad (i\in\{1,\ldots,K\}), \tag{6.3}$$

which can be interpreted as follows: Given the optimal K-clustering, the clustering point associated with each cluster must be the centroid of the cluster.

Given K clustering points, for a generic feature point $\boldsymbol{P}_j$ $(j\in\{1,\ldots,N\})$, denote

$$\boldsymbol{m}_{j'} \equiv \text{argmin}_{\boldsymbol{m}_{j'}\in\{\boldsymbol{m}_1,\ldots,\boldsymbol{m}_K\}}\|\boldsymbol{P}_j-\boldsymbol{m}_{j'}\|_2. \tag{6.4}$$

The objective function specified in (6.1) also satisfies the inequality

$$\sum_{i=1}^{K}\sum_{\boldsymbol{P}_j\in\boldsymbol{K}_i}\|\boldsymbol{P}_j-\boldsymbol{m}_i\|_2^2 \geqslant \sum_{i=1}^{K}\sum_{\boldsymbol{P}_j\in\boldsymbol{K}_i}\|\boldsymbol{P}_j-\boldsymbol{m}_{j'}\|_2^2. \tag{6.5}$$

The equality of (6.5) holds if and only if

$$\boldsymbol{m}_{j'}=\boldsymbol{m}_i \quad (j'\in\{1,\ldots,N\};i\in\{1,\ldots,K\}), \tag{6.6}$$

which can be interpreted as follows: Given the optimal K-clustering, each feature point must be assigned to the closest clustering point and to the cluster associated with the closest clustering point.

The conditions (6.3) and (6.6) together form the theoretical conclusion: *Given the optimal K-clustering, the clustering point associated with each cluster must be the centroid of the cluster, and each feature point must be assigned to the closest cluster centroid as well as the associated cluster.* This theoretical conclusion underlies the K-means algorithm, as follows.

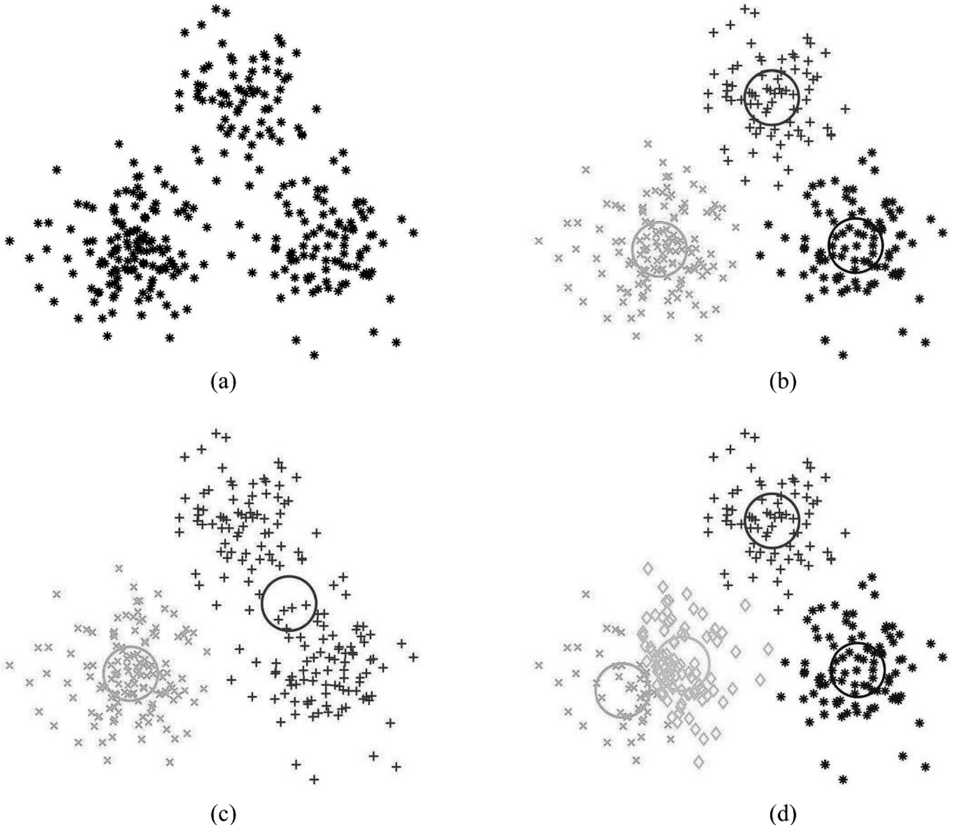
Algorithm: K-Means

```
Select $K$ feature points randomly as cluster centroids
WHILE True
    FOR EACH feature point
        Assign feature point to the closest cluster (centroid)
    END
    FOR EACH cluster
        Update the cluster centroid
    END
    IF $K$-clustering unchanged
        BREAK
    END
END
```

In above pseudo code, there is a loop traversing the feature points and another loop traversing the clusters. In the second loop, it seems that the update of each cluster centroid involves traversal of the feature points assigned to the cluster. In fact, such seemingly needed traversal of feature points in the second loop can be totally saved because they can be merged into the traversal of feature points in the first loop. Matlab code for K-means clustering is given as follows.

KMeans.m

```
%% K-means clustering
% Kc : Indexing of feature points clustered
% Km : Cluster centroids
% Kv : Summed cluster variances (moment d’inertie)
function [Kc, Km, Kv] = KMeans(p, K)
    N = size(p,2); Kc = zeros(N,K);
    % Initialize via random selection of K points
    i = ceil(N*rand(1,K)); Km = [p(:,i);zeros(1,K)];
    % Iterative refinement of K-clustering
    while (1)
        Kmo=Km(1:end-1,:); Km(:,:)=0; Kco=Kc; Kc(:,:)=0;
        for k=1:N
            d = sum((repmat(p(:,k),1,K)-Kmo).^2,1);
            i = find(d==min(d)); i=i(1); Kc(k,i)=1;
            Km(:,i) = Km(:,i) + [p(:,k);1];
        end
        for k=1:K
            if (Km(end,k)==0) continue; end
            Km(1:end-1,k) = Km(1:end-1,k)/Km(end,k);
        end
        if(max(max(abs(Kc-Kco))) == 0) break; end
    end
    Kv = 0;
    for k=1:K
        i = find(Kc(:,k)==1);
        d = p(:,i)-repmat(Km(1:end-1,k),1,length(i));
        Kv = Kv + sum(sum(d.^2));
    end
end
```



(a) (b) (c) (d)

Fig. 6.2 K-means clustering **a** original feature points; **b** optimal 3-clustering; **c** optimal 2-clustering; **d** optimal 4-clustering

An example of K-means clustering is demonstrated in Fig. 6.2, where the sub-figure(a) illustrates an original set of two-dimensional feature points. Intuitively speaking, the feature points seem to form three clusters, so apply the K-means method with $K = 3$ to cluster the feature points. The optimal 3-clustering is illustrated in Fig. 6.2b, where feature points assigned to the three clusters are marked by the shapes "+", "X", and "*" respectively. The centroids of the clusters are indicated by the circles. The illustrated 3-clustering effect is rather consistent with intuition.

An often claimed inconvenience of the K-means method consists in the need to specify the number of clusters, i.e. the number K, whereas a priori knowledge of K may be or even tends to be unavailable in practical applications. On one hand, the claimed inconvenience indeed exists. On the other hand, one had better not exaggerate the claimed inconvenience, mainly for two reasons: First, it is sometimes unnecessary to scrutinize the so-called objective number of clusters, and an empirical choice from a rough range of K is already proper enough to satisfy some practical requirements.

Second, the spirit of multi-scale analysis can be incorporated into the K-means method for flexible clustering. More specifically, instead of applying the K-means method with a fixed K, one may apply the K-means method with a range of K respectively to analyse how variation of K influences clustering effects and make a clustering decision adaptively according to such analysis.

For example, given the same set of two-dimensional feature points illustrated in Fig. 6.2a, apply the K-means method with $K = 2$, $K = 3$, and $K = 4$ respectively to cluster the feature points. The optimal 2-clustering, the optimal 3-clustering, and the optimal 4-clustering are illustrated in Fig. 6.2b, c, d respectively.

As shown in Fig. 6.2, from the optimal 2-clustering to the optimal 3-clustering, the "+" cluster in the optimal 2-clustering is divided into two smaller clusters which are apart considerably on one hand and are rather self-concentrated respectively on the other hand. In other words, the divided two smaller clusters are rather distinct from each other and had better be indeed treated as two clusters instead of a merged cluster. So the optimal 3-clustering seems to be more natural than the optimal 2-clustering. As also shown in Fig. 6.2, from the optimal 3-clustering to the optimal 4-clustering, the "X" cluster in the optimal 3-clustering is divided into two smaller clusters which are not so distinct from each other. Compared with the optimal 3-clustering, it seems to be not worth having one more cluster in the optimal 4-clustering. Therefore, one may conclude that the optimal 3-clustering tends to be a proper choice for the given example of feature points .[2]

6.1.3 Mean Shift

The **mean shift** method [5–7] is another representative clustering method, which is based on the probabilistic methodology of conceptualizing feature points as samples of a probability density function over feature space and on the cognition that the probability density function value at a point can be fairly estimated with sample values around it, namely via *nonparametric kernel estimation* [8, 9].

Given a set of N sample points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_N$ in the d-dimensional feature space, the d-dimensional multivariate *kernel density estimator* is formalized as

$$\hat{f}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} K(\boldsymbol{x} - \boldsymbol{x}_i), \tag{6.7}$$

where the d-variate kernel $K(\boldsymbol{x})$ satisfies the following conditions:

- *Normalization condition* : $\int_{\mathbb{R}^d} K(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$.
- *Concentration condition* : $\lim_{\|\boldsymbol{x}\|\to\infty} \|\boldsymbol{x}\|^d K(\boldsymbol{x}) = 0$.
- *Unbiasedness condition* : $\int_{\mathbb{R}^d} \boldsymbol{x}K(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \boldsymbol{0}$.
- *Uniformity condition* : $\int_{\mathbb{R}^d} \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}K(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = c_K\boldsymbol{I}$ (c_K is a constant).

An important kernel form for choice is the *radial symmetric kernel* form

$$K(\boldsymbol{x}) \propto \bar{K}(\|\boldsymbol{x}\|_2^2) = \bar{K}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}), \tag{6.8}$$

where the function $\bar{K}(\cdot)$ is called the *profile* of the kernel. Besides, the Euclidean distance based norm square $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ in (6.8) can be generalized to the Mahalanobis distance based norm square $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{x}$ (see Sect. 3.2.3 in Chap. 3). This can be treated as if one takes a coordinates transform and substitutes $\tilde{\boldsymbol{x}} = \boldsymbol{M}^{-1/2}\boldsymbol{x}$ for $\boldsymbol{x}$ in the kernel $K(\boldsymbol{x})$. To guarantee the normalization condition , one must have

$$K(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = K(\tilde{\boldsymbol{x}})\mathrm{d}\tilde{\boldsymbol{x}} = K(\tilde{\boldsymbol{x}})|\boldsymbol{M}^{-1/2}|\mathrm{d}\boldsymbol{x} \Longleftrightarrow K(\boldsymbol{x}) = \frac{1}{\sqrt{|\boldsymbol{M}|}}K(\tilde{\boldsymbol{x}})$$

which implies a generalized form of (6.8):

$$K(\boldsymbol{x}) = \tfrac{1}{\sqrt{|\boldsymbol{M}|}} K(\boldsymbol{M}^{-1/2}\boldsymbol{x}) \propto \tfrac{1}{\sqrt{|\boldsymbol{M}|}} \bar{K}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{x}) \tag{6.9}$$

Substitute (6.9) into (6.7) and obtain a special form of the kernel density estimator

$$\begin{aligned} \hat{f}(\boldsymbol{x}) = & \quad \frac{1}{N\sqrt{|\boldsymbol{M}|}} \sum_{i=1}^{N} K(\boldsymbol{M}^{-1/2}(\boldsymbol{x} - \boldsymbol{x}_i)) \\ \propto & \quad \frac{1}{N\sqrt{|\boldsymbol{M}|}} \sum_{i=1}^{N} \bar{K}((\boldsymbol{x} - \boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{M}^{-1} (\boldsymbol{x} - \boldsymbol{x}_i)). \end{aligned} \tag{6.10}$$

In practice, the matrix $\boldsymbol{M}$ is conventionally set to $h^2\boldsymbol{I}$ for processing convenience,[3] based on which (6.10) is reduced to

$$\hat{f}(\boldsymbol{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right) \propto \frac{1}{Nh^d} \sum_{i=1}^{N} \bar{K}\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right). \tag{6.11}$$

Compute the analytical expression of the gradient of $\hat{f}(\boldsymbol{x})$ in (6.11) as

$$\begin{aligned} \nabla \hat{f}(\boldsymbol{x}) & \propto \frac{2}{Nh^{d+2}} \sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)(\boldsymbol{x} - \boldsymbol{x}_i) \\ & = \frac{2}{Nh^{d+2}} \left[\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)\right] \left[\boldsymbol{x} - \frac{\sum_{i=1}^{N} \boldsymbol{x}_i \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)}{\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)}\right] \end{aligned} \tag{6.12}$$

which takes zero at

$$\boldsymbol{x} = \frac{\sum_{i=1}^{N} \boldsymbol{x}_i \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)}{\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right\|_2^2\right)}. \tag{6.13}$$

The nonlinear equation (6.13) in terms of $\boldsymbol{x}$ can be solved iteratively via

$$\boldsymbol{x}_{t+1} = \frac{\sum_{i=1}^{N} \boldsymbol{x}_i \bar{K}'\left(\left\|\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right\|_2^2\right)}{\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right\|_2^2\right)}, \tag{6.14}$$

if the profile function $\bar{K}(\cdot)$ is convex and decreases monotonically. By "can be solved" the author means that given an initial $\boldsymbol{x}$, the iteration via (6.14) will make $\boldsymbol{x}$ converge to a solution of the nonlinear equation (6.13), or equivalently make $\hat{f}(\boldsymbol{x})$ converge to an extremum (more specifically, a maximum), i.e. a *mode* in feature space.

Proof The profile function $\bar{K}(\cdot)$ decreasing monotonically implies that $-\bar{K}^{'}(\cdot) \geqslant 0$. The profile function $\bar{K}(\cdot)$ being convex implies that

$$\forall y_1, y_2, \quad \bar{K}(y_2) - \bar{K}(y_1) \geqslant \bar{K}^{'}(y_1)(y_2 - y_1) = [-\bar{K}^{'}(y_1)](y_1 - y_2).$$

So

$$
\begin{aligned}
& \hat{f}(\boldsymbol{x}_{t+1}) - \hat{f}(\boldsymbol{x}_t) \\
\propto\ & \sum_{i=1}^{N} \bar{K}\left(\left\|\frac{\boldsymbol{x}_{t+1}-\boldsymbol{x}_i}{h}\right\|_2^2\right) - \sum_{i=1}^{N} \bar{K}\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right) \\
\geqslant\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right]\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2 - \left\|\frac{\boldsymbol{x}_{t+1}-\boldsymbol{x}_i}{h}\right\|_2^2\right) \\
\propto\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right][\boldsymbol{x}_t^{\mathrm{T}}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_{t+1} + 2\boldsymbol{x}_i^{\mathrm{T}}(\boldsymbol{x}_{t+1}-\boldsymbol{x}_t)] \\
=\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right](\boldsymbol{x}_t^{\mathrm{T}}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_{t+1}) + \\
& 2\left[-\sum_{i=1}^{N} \boldsymbol{x}_i^{\mathrm{T}}\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right](\boldsymbol{x}_{t+1}-\boldsymbol{x}_t) \\
=\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right](\boldsymbol{x}_t^{\mathrm{T}}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_{t+1}) + \\
& 2\left[-\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right]\boldsymbol{x}_{t+1}^{\mathrm{T}}(\boldsymbol{x}_{t+1}-\boldsymbol{x}_t) \quad (\text{using}(6.14)) \\
=\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right](\boldsymbol{x}_t^{\mathrm{T}}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_{t+1} + 2\boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_{t+1} - 2\boldsymbol{x}_{t+1}^{\mathrm{T}}\boldsymbol{x}_t) \\
=\ & \sum_{i=1}^{N} \left[-\bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)\right](\boldsymbol{x}_t - \boldsymbol{x}_{t+1})^{\mathrm{T}}(\boldsymbol{x}_t - \boldsymbol{x}_{t+1}) \geqslant 0,
\end{aligned}
$$

which implies that the bounded sequence $\{\hat{f}(\boldsymbol{x}_t)\}$ is monotonically increasing and hence converges to a limit corresponding to a maximum in feature space. The proof is done.□

The iteration from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ described by (6.14), from the geometry perspective, is like one shifts $\boldsymbol{x}_t$ by

$$
\frac{\sum_{i=1}^{N} \boldsymbol{x}_i \bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)}{\sum_{i=1}^{N} \bar{K}'\left(\left\|\frac{\boldsymbol{x}_{t}-\boldsymbol{x}_i}{h}\right\|_2^2\right)} - \boldsymbol{x}_t, \tag{6.15}
$$

which is called the **mean shift** .

By so far, the author has been describing the profile function $\bar{K}(\cdot)$ abstractly. For selection of a concrete profile function $\bar{K}(\cdot)$ that needs to satisfy the convexity and monotonically-decreasing conditions, adopt the triangular kernel (instead of the normal kernel) recommended by [7]

$$\bar{K}(y)=\begin{cases}1-y, & 0\leqslant y\leqslant 1\\ 0, & y>1\end{cases} \tag{6.16}$$

So $-\bar{K}'(y)$ is the uniform kernel

$$-\bar{K}'(y)=\begin{cases}1, & 0\leqslant y\leqslant 1\\ 0, & y>1\end{cases} \tag{6.17}$$

Substitute (6.17) into (6.14) and obtain

$$\boldsymbol{x}_{t+1}=\frac{\sum\limits_{\boldsymbol{x}_i\in\{\boldsymbol{x}|\|\boldsymbol{x}-\boldsymbol{x}_t\|_2\leqslant h\}}\boldsymbol{x}_i}{|\{\boldsymbol{x}|\|\boldsymbol{x}-\boldsymbol{x}_t\|_2\leqslant h\}|} \tag{6.18}$$

where $|\{\boldsymbol{x}|\|\boldsymbol{x}-\boldsymbol{x}_t\|_2\leqslant h\}|$ denotes the cardinality of the set $\{\boldsymbol{x}|\|\boldsymbol{x}-\boldsymbol{x}_t\|_2\leqslant h\}$. Matlab code for mean shift clustering of feature points is given as follows.

MeanShift.m

```
%% Mean shift clustering of feature points
% pm : [points; modes; clustering indices]
function pm = MeanShift(p, h)
    [dn,N] = size(p); pm = [p;p;ones(1,N)]; err=0.001;
    for k=1:N
        pk = p(:,k);
        while(1)
            pold = pk;
            % Find feature points within kernel bandwidth
            % Computationally consuming
            d = p-repmat(pk,1,N); d = sum(d.^2,1);
            i = find(d<=h^2);
            % Mean shift
            pk = mean(p(:,i),2);
            if(sum(abs(pk-pold))<err) break; end
        end
        pm(dn+1:dn+dn,k) = pk;
    end
    % Refine clustering
    ms = zeros(2*dn,0); % [x1min; ... xnmin; x1max; ... xnmax]
    for k=1:N
        cflg = -1; pk = pm(dn+1:dn+dn,k);
        for j=1:size(ms,2)
            if (pk>=ms(1:dn,j)-100*err & pk<=ms(dn+1:dn+dn,j)
    +100*err)
                ms(1:dn,j) = min(ms(1:dn,j),pk);
                ms(dn+1:dn+dn,j) = max(ms(dn+1:dn+dn,j),pk);
                cflg = j; pm(end,k) = cflg; break;
            end
        end
        if (cflg<0) ms=[ms,[pk;pk]]; pm(end,k)=size(ms,2); end
    end
end
```

Still take the example of feature points illustrated in Fig. 6.2a. Apply the mean shift method (using above Matlab script **MeanShift.m**) to perform clustering of the feature points, the effect of which is demonstrated in Fig. 6.3a. The feature points are assigned to three clusters marked by the shapes “+”, “X”, and “*” respectively. The clustering result is rather consistent with the intuition that the feature points seem to form three clusters. Unlike the K-means method that specifies the cluster number explicitly, the mean shift method is exempt from explicit specification of the cluster number.

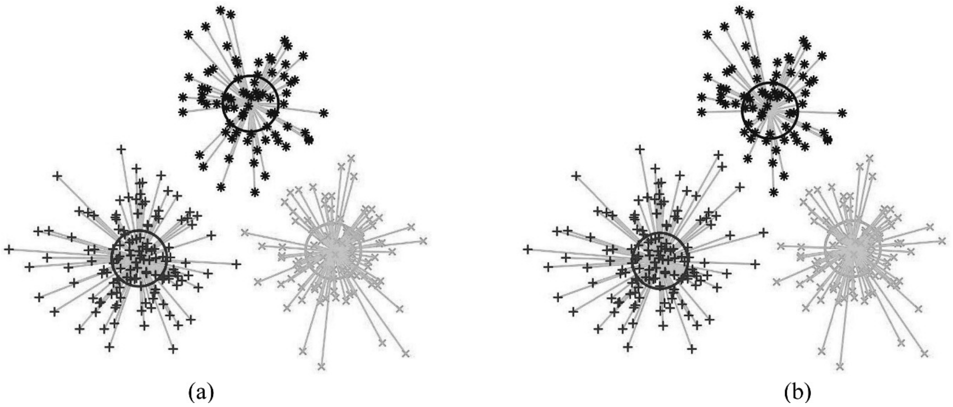


Fig. 6.3 Mean shift clustering **a** direct implementation; **b** grid sampling based implementation

It is worth noting that tuning of the kernel bandwidth h will influence the effect of mean shift clustering. When a smaller kernel bandwidth h is adopted, the feature points tend to converge into more clusters. When a larger kernel bandwidth h is adopted, the feature points tend to converge into less clusters. It is when one selects the kernel bandwidth h in certain suitable range that the three clusters of feature points can be formed, as demonstrated in Fig. 6.3a.

It is also worth noting that due to influences of ad hoc factors such as numeric computation errors and kernel modelling errors, the mean shift method (6.15) in practice is unlikely to make feature points associated with the same mode in feature space actually converge to exactly the same position of the mode, though it should do so theoretically. In the demonstrated Matlab script **MeanShift.m**, a final step of “refine clustering” is involved to cluster close mode positions into the same mode.

Kernel bandwidth h

The K-means method involves one parameter namely the cluster number to tune, whereas the mean shift method also involves one parameter namely the kernel bandwidth h to tune. Both methods seem to have no essential difference only in terms of the number of parameters to tune. However, from the practice perspective, tuning of the kernel bandwidth h is much more natural than tuning of the cluster number. The reasons are three-folds: First, a priori knowledge of the cluster number or even of a rough range of the cluster number is indeed unavailable for many practical applications. Tuning of the cluster number is after all undesirable and inconvenient despite the possibility of multi-scale analysis. Second, fair a priori knowledge of evaluating closeness and apartness among feature points, which is reflected by the kernel bandwidth h, can usually be acquired in practice. Third, tuning of the kernel bandwidth h is comparatively intuitive for analysing how the clustering effect would vary accordingly.

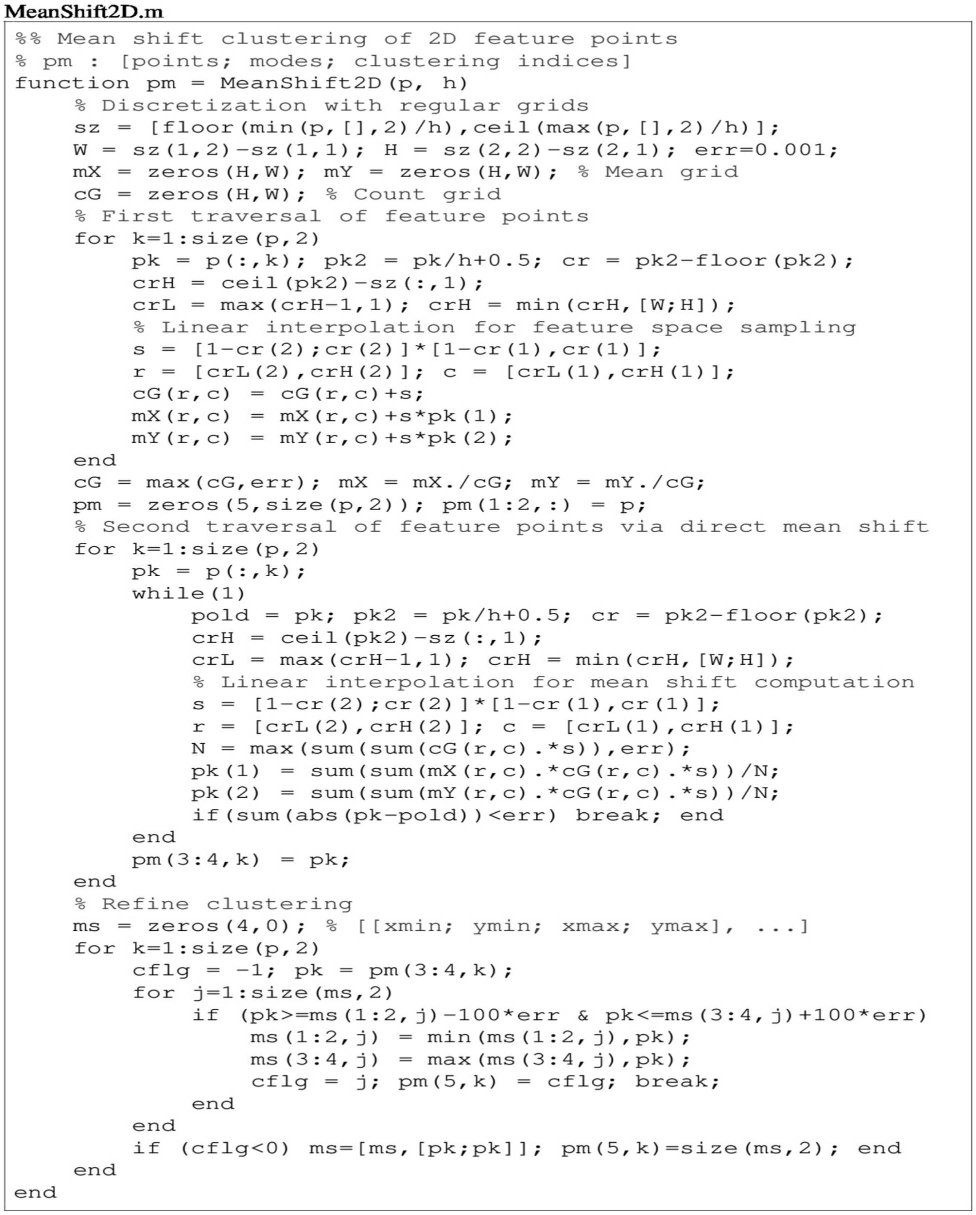
By analogy, suppose one intends to assign guests participating in a banquet to different groups. The first way, i.e. the K-means way of assigning guests is like one prescribes certain imagined or guessed total number of groups at the very beginning and then rigidly “crams” guests to the fixed number of groups. The second way, i.e. the mean shift way of assigning guests is like one first establishes a criterion to evaluate closeness and apartness among guests (for example, one may regard guests sitting at the same table as having close relationship, or regard guests staying in the same meeting room as having close relationship, according to concrete needs) and then lets guests naturally form different groups according to the established criterion. The author believes that most readers tend to regard the second way of assigning guests as being more intuitively conceivable, more convenient, more tractable, and more effective than the first way and hence tend to adopt the second way.

Grid sampling-interpolation based implementation

The Matlab script **MeanShift.m** gives a direct or literal implementation of the mean shift method, which is inefficient at all due to the computationally consuming step of finding “close” feature points at each mean shift iteration. One way for computational efficiency enhancement is to take advantage of advanced data structures involving closest point problems in computation geometry. Another way for computational efficiency enhancement, especially suitable to the simple mean shift formalism (6.18), is to take advantage of feature space sampling.

The basic idea is: discretize feature space into regular grids, perform sampling of feature points on grids, and pre-compute the mean of feature points in each grid. After sampling on grids and pre-computation of grid means, the feature points, starting from their original positions in feature space, are conveniently made converge to mode positions via mean shift , with the help of grid data prepared beforehand. For convergence of a feature point via grid sampling based mean shift , at each mean shift iteration, quickly locate the grid that contains the feature point currently and move the feature point to the grid mean which can be retrieved immediately.

When putting the basic idea into practice, one had better bear in mind that*interpolation* (say *linear interpolation*) is important to grid sampling, grid mean pre-computation, and grid mean retrieval at each mean shift iteration, in terms of guaranteeing continuity of the prepared grid data. Matlab code for mean shift clustering of two-dimensional feature points, which adopts grid sampling based implementation, is given as follows.

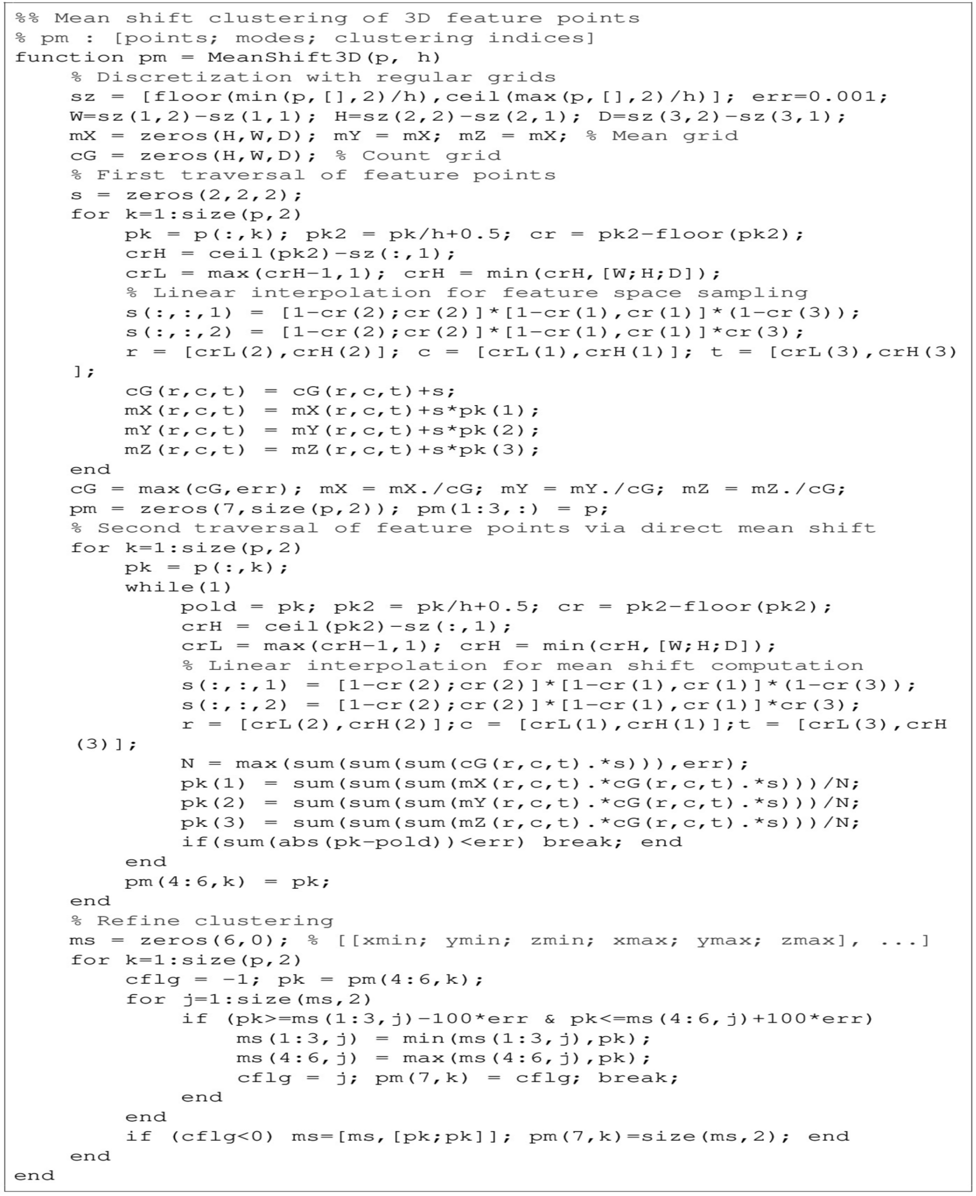


MeanShift2D.m

```
%% Mean shift clustering of 2D feature points
% pm : [points; modes; clustering indices]
function pm = MeanShift2D(p, h)
    % Discretization with regular grids
    sz = [floor(min(p,[],2)/h),ceil(max(p,[],2)/h)];
    W = sz(1,2)-sz(1,1); H = sz(2,2)-sz(2,1); err=0.001;
    mX = zeros(H,W); mY = zeros(H,W); % Mean grid
    cG = zeros(H,W); % Count grid
    % First traversal of feature points
    for k=1:size(p,2)
        pk = p(:,k); pk2 = pk/h+0.5; cr = pk2-floor(pk2);
        crH = ceil(pk2)-sz(:,1);
        crL = max(crH-1,1); crH = min(crH,[W;H]);
        % Linear interpolation for feature space sampling
        s = [1-cr(2);cr(2)]*[1-cr(1),cr(1)];
        r = [crL(2),crH(2)]; c = [crL(1),crH(1)];
        cG(r,c) = cG(r,c)+s;
        mX(r,c) = mX(r,c)+s*pk(1);
        mY(r,c) = mY(r,c)+s*pk(2);
    end
    cG = max(cG,err); mX = mX./cG; mY = mY./cG;
    pm = zeros(5,size(p,2)); pm(1:2,:) = p;
    % Second traversal of feature points via direct mean shift
    for k=1:size(p,2)
        pk = p(:,k);
        while(1)
            pold = pk; pk2 = pk/h+0.5; cr = pk2-floor(pk2);
            crH = ceil(pk2)-sz(:,1);
            crL = max(crH-1,1); crH = min(crH,[W;H]);
            % Linear interpolation for mean shift computation
            s = [1-cr(2);cr(2)]*[1-cr(1),cr(1)];
            r = [crL(2),crH(2)]; c = [crL(1),crH(1)];
            N = max(sum(sum(cG(r,c).*s)),err);
            pk(1) = sum(sum(mX(r,c).*cG(r,c).*s))/N;
            pk(2) = sum(sum(mY(r,c).*cG(r,c).*s))/N;
            if(sum(abs(pk-pold))<err) break; end
        end
        pm(3:4,k) = pk;
    end
    % Refine clustering
    ms = zeros(4,0); % [[xmin; ymin; xmax; ymax], ...]
    for k=1:size(p,2)
        cflg = -1; pk = pm(3:4,k);
        for j=1:size(ms,2)
            if (pk>=ms(1:2,j)-100*err & pk<=ms(3:4,j)+100*err)
                ms(1:2,j) = min(ms(1:2,j),pk);
                ms(3:4,j) = max(ms(3:4,j),pk);
                cflg = j; pm(5,k) = cflg; break;
            end
        end
        if (cflg<0) ms=[ms,[pk;pk]]; pm(5,k)=size(ms,2); end
    end
end
```

Matlab code for mean shift clustering of three-dimensional feature points, which adopts grid sampling based implementation as well, is given as follows. The three-dimensional feature space oriented implementation given in the Matlab script **MeanShift3D.m** is a simple extension of the two-dimensional feature space oriented implementation given in the Matlab script **MeanShift2D.m**.

MeanShift3D.m



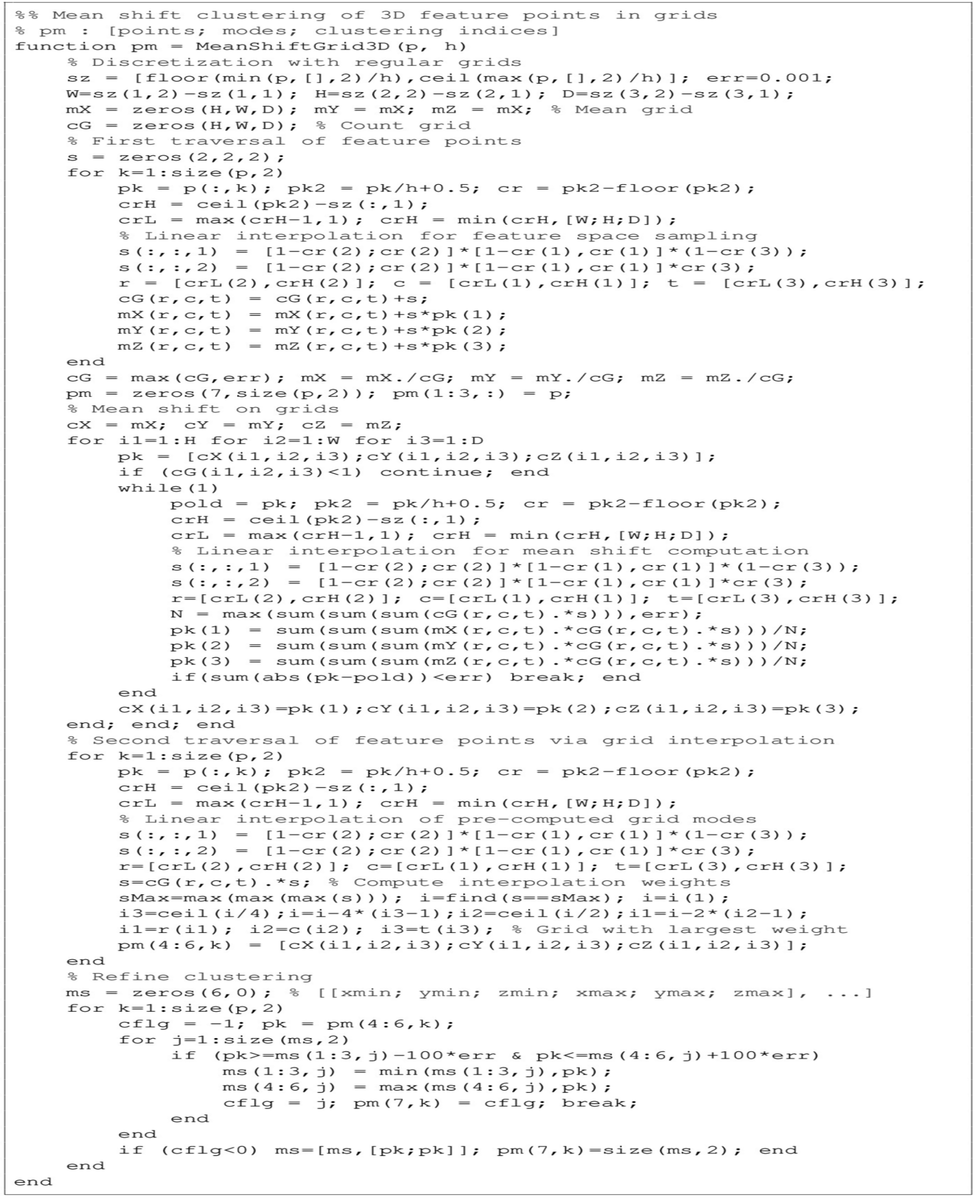
```
%% Mean shift clustering of 3D feature points
% pm : [points; modes; clustering indices]
function pm = MeanShift3D(p, h)
    % Discretization with regular grids
    sz = [floor(min(p,[],2)/h),ceil(max(p,[],2)/h)]; err=0.001;
    W=sz(1,2)-sz(1,1); H=sz(2,2)-sz(2,1); D=sz(3,2)-sz(3,1);
    mX = zeros(H,W,D); mY = mX; mZ = mX; % Mean grid
    cG = zeros(H,W,D); % Count grid
    % First traversal of feature points
    s = zeros(2,2,2);
    for k=1:size(p,2)
        pk = p(:,k); pk2 = pk/h+0.5; cr = pk2-floor(pk2);
        crH = ceil(pk2)-sz(:,1);
        crL = max(crH-1,1); crH = min(crH,[W;H;D]);
        % Linear interpolation for feature space sampling
        s(:,:,1) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*(1-cr(3));
        s(:,:,2) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*cr(3);
        r = [crL(2),crH(2)]; c = [crL(1),crH(1)]; t = [crL(3),crH(3)
    ];
        cG(r,c,t) = cG(r,c,t)+s;
        mX(r,c,t) = mX(r,c,t)+s*pk(1);
        mY(r,c,t) = mY(r,c,t)+s*pk(2);
        mZ(r,c,t) = mZ(r,c,t)+s*pk(3);
    end
    cG = max(cG,err); mX = mX./cG; mY = mY./cG; mZ = mZ./cG;
    pm = zeros(7,size(p,2)); pm(1:3,:) = p;
    % Second traversal of feature points via direct mean shift
    for k=1:size(p,2)
        pk = p(:,k);
        while(1)
            pold = pk; pk2 = pk/h+0.5; cr = pk2-floor(pk2);
            crH = ceil(pk2)-sz(:,1);
            crL = max(crH-1,1); crH = min(crH,[W;H;D]);
            % Linear interpolation for mean shift computation
            s(:,:,1) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*(1-cr(3));
            s(:,:,2) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*cr(3);
            r = [crL(2),crH(2)];c = [crL(1),crH(1)];t = [crL(3),crH
     (3)];
            N = max(sum(sum(sum(cG(r,c,t).*s))),err);
            pk(1) = sum(sum(sum(mX(r,c,t).*cG(r,c,t).*s)))/N;
            pk(2) = sum(sum(sum(mY(r,c,t).*cG(r,c,t).*s)))/N;
            pk(3) = sum(sum(sum(mZ(r,c,t).*cG(r,c,t).*s)))/N;
            if(sum(abs(pk-pold))<err) break; end
        end
        pm(4:6,k) = pk;
    end
    % Refine clustering
    ms = zeros(6,0); % [[xmin; ymin; zmin; xmax; ymax; zmax], ...]
    for k=1:size(p,2)
        cflg = -1; pk = pm(4:6,k);
        for j=1:size(ms,2)
            if (pk>=ms(1:3,j)-100*err & pk<=ms(4:6,j)+100*err)
                ms(1:3,j) = min(ms(1:3,j),pk);
                ms(4:6,j) = max(ms(4:6,j),pk);
                cflg = j; pm(7,k) = cflg; break;
            end
        end
        if (cflg<0) ms=[ms,[pk;pk]]; pm(7,k)=size(ms,2); end
    end
end
```

If there are a large amount of feature points (for example, in the application of image segmentation which will be demonstrated latter), the grid data prepared beforehand via grid sampling and grid mean pre-computation cannot guarantee computation efficiency by themselves alone, because convergence of all feature points via grid sampling based mean shift is still computationally consuming. Therefore, to enhance computational efficiency, besides preparing grid data in advance via grid sampling and grid mean pre-computation on one hand, one also needs on the other hand to improve the way of making feature points converge to their corresponding modes in feature space.

For this purpose, one can take advantage of already prepared grid data. More specifically, only apply mean shift to make grid means converge to feature modes. Then for any feature point, compute its linear interpolation weights (considering numbers of feature points in grids) distributed over the grids surrounding it, retrieve the grid with the largest weight, and assign the feature mode associated with the retrieved grid to the feature point. Such technique is referred to as *grid interpolation*, whereas the previously presented technique of grid sampling and grid mean computation is referred to as *grid sampling*. Combination of both techniques is referred to as *grid sampling-interpolation*.

The Matlab scripts **MeanShift2D.m** and **MeanShift3D.m** adopt implementation based on grid sampling only. Matlab code for mean shift clustering of three-dimensional feature points, which adopts grid sampling-interpolation based implementation, i.e. both grid sampling and grid interpolation based implementation, is given as follows.

MeanShiftGrid3D.m



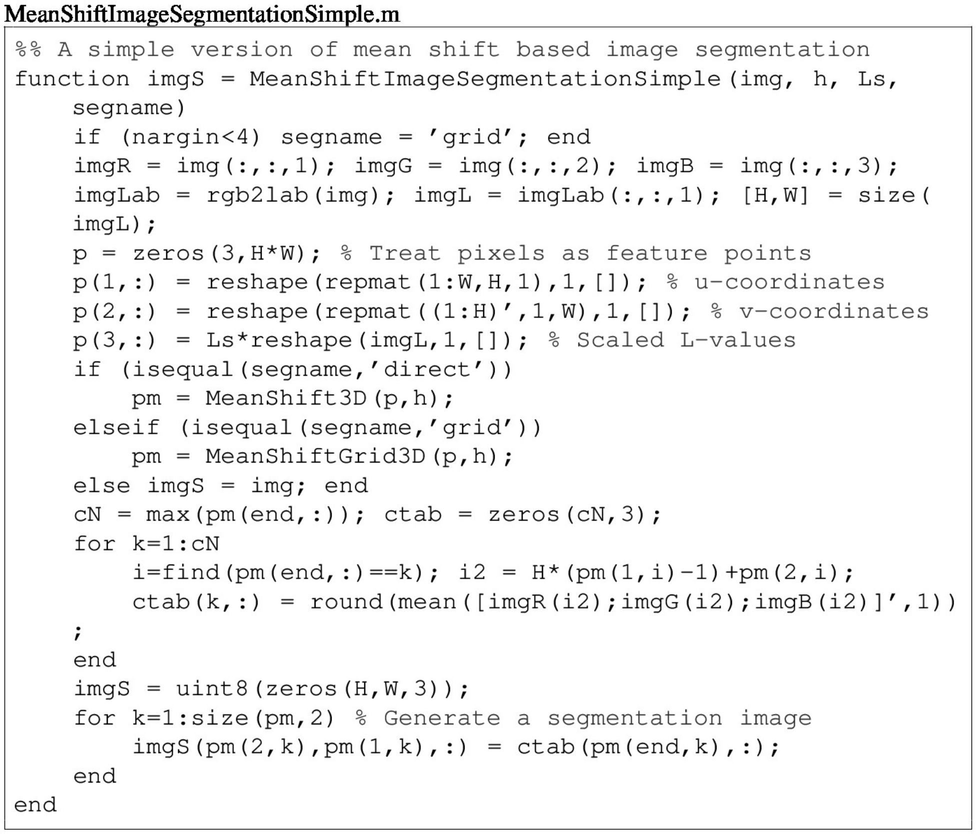
```
%% Mean shift clustering of 3D feature points in grids
% pm : [points; modes; clustering indices]
function pm = MeanShiftGrid3D(p, h)
    % Discretization with regular grids
    sz = [floor(min(p,[],2)/h),ceil(max(p,[],2)/h)]; err=0.001;
    W=sz(1,2)-sz(1,1); H=sz(2,2)-sz(2,1); D=sz(3,2)-sz(3,1);
    mX = zeros(H,W,D); mY = mX; mZ = mX; % Mean grid
    cG = zeros(H,W,D); % Count grid
    % First traversal of feature points
    s = zeros(2,2,2);
    for k=1:size(p,2)
        pk = p(:,k); pk2 = pk/h+0.5; cr = pk2-floor(pk2);
        crH = ceil(pk2)-sz(:,1);
        crL = max(crH-1,1); crH = min(crH,[W;H;D]);
        % Linear interpolation for feature space sampling
        s(:,:,1) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*(1-cr(3));
        s(:,:,2) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*cr(3);
        r = [crL(2),crH(2)]; c = [crL(1),crH(1)]; t = [crL(3),crH(3)];
        cG(r,c,t) = cG(r,c,t)+s;
        mX(r,c,t) = mX(r,c,t)+s*pk(1);
        mY(r,c,t) = mY(r,c,t)+s*pk(2);
        mZ(r,c,t) = mZ(r,c,t)+s*pk(3);
    end
    cG = max(cG,err); mX = mX./cG; mY = mY./cG; mZ = mZ./cG;
    pm = zeros(7,size(p,2)); pm(1:3,:) = p;
    % Mean shift on grids
    cX = mX; cY = mY; cZ = mZ;
    for i1=1:H for i2=1:W for i3=1:D
        pk = [cX(i1,i2,i3);cY(i1,i2,i3);cZ(i1,i2,i3)];
        if (cG(i1,i2,i3)<1) continue; end
        while(1)
            pold = pk; pk2 = pk/h+0.5; cr = pk2-floor(pk2);
            crH = ceil(pk2)-sz(:,1);
            crL = max(crH-1,1); crH = min(crH,[W;H;D]);
            % Linear interpolation for mean shift computation
            s(:,:,1) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*(1-cr(3));
            s(:,:,2) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*cr(3);
            r=[crL(2),crH(2)]; c=[crL(1),crH(1)]; t=[crL(3),crH(3)];
            N = max(sum(sum(sum(cG(r,c,t).*s))),err);
            pk(1) = sum(sum(sum(mX(r,c,t).*cG(r,c,t).*s)))/N;
            pk(2) = sum(sum(sum(mY(r,c,t).*cG(r,c,t).*s)))/N;
            pk(3) = sum(sum(sum(mZ(r,c,t).*cG(r,c,t).*s)))/N;
            if(sum(abs(pk-pold))<err) break; end
        end
        cX(i1,i2,i3)=pk(1);cY(i1,i2,i3)=pk(2);cZ(i1,i2,i3)=pk(3);
    end; end; end
    % Second traversal of feature points via grid interpolation
    for k=1:size(p,2)
        pk = p(:,k); pk2 = pk/h+0.5; cr = pk2-floor(pk2);
        crH = ceil(pk2)-sz(:,1);
        crL = max(crH-1,1); crH = min(crH,[W;H;D]);
        % Linear interpolation of pre-computed grid modes
        s(:,:,1) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*(1-cr(3));
        s(:,:,2) = [1-cr(2);cr(2)]*[1-cr(1),cr(1)]*cr(3);
        r=[crL(2),crH(2)]; c=[crL(1),crH(1)]; t=[crL(3),crH(3)];
        s=cG(r,c,t).*s; % Compute interpolation weights
        sMax=max(max(max(s))); i=find(s==sMax); i=i(1);
        i3=ceil(i/4);i=i-4*(i3-1);i2=ceil(i/2);i1=i-2*(i2-1);
        i1=r(i1); i2=c(i2); i3=t(i3); % Grid with largest weight
        pm(4:6,k) = [cX(i1,i2,i3);cY(i1,i2,i3);cZ(i1,i2,i3)];
    end
    % Refine clustering
    ms = zeros(6,0); % [[xmin; ymin; zmin; xmax; ymax; zmax], ...]
    for k=1:size(p,2)
        cflg = -1; pk = pm(4:6,k);
        for j=1:size(ms,2)
            if (pk>=ms(1:3,j)-100*err & pk<=ms(4:6,j)+100*err)
                ms(1:3,j) = min(ms(1:3,j),pk);
                ms(4:6,j) = max(ms(4:6,j),pk);
                cflg = j; pm(7,k) = cflg; break;
            end
        end
        if (cflg<0) ms=[ms,[pk;pk]]; pm(7,k)=size(ms,2); end
    end
end
```

Although the grid sampling-interpolation technique seems to be “coarse”, as will be demonstrated later with image segmentation examples, it usually achieves almost the same effect as that of performing mean shift on all feature points to make them converge to feature.

modes **Mean shift application to image segmentation**

An interesting example of mean shift application is image segmentation . Here, the author has no intention to scrutinize “good” techniques that enhance the image segmentation performance, but only intends to demonstrate how the mean shift method can be applied in practice. So the author intentionally provides a simple version of mean shift based image segmentation method, for sake of minimizing ad hoc considerations [4] concerning image segmentation only but having no relation to the mean shift method itself.

Explanations hover over the simple method of image segmentation . Transform the original image into the L*a*b space [12] and take only the L-value for each pixel, so image pixels together with their associated L-values form a dense set of three-dimensional feature points. Then perform mean shift clustering of them for image segmentation purpose. Matlab code for the simple version of mean shift based image segmentation is given as follows.



MeanShiftImageSegmentationSimple.m

```
%% A simple version of mean shift based image segmentation
function imgS = MeanShiftImageSegmentationSimple(img, h, Ls,
    segname)
    if (nargin<4) segname = ’grid’; end
    imgR = img(:,:,1); imgG = img(:,:,2); imgB = img(:,:,3);
    imgLab = rgb2lab(img); imgL = imgLab(:,:,1); [H,W] = size(
    imgL);
    p = zeros(3,H*W); % Treat pixels as feature points
    p(1,:) = reshape(repmat(1:W,H,1),1,[]); % u-coordinates
    p(2,:) = reshape(repmat((1:H)’,1,W),1,[]); % v-coordinates
    p(3,:) = Ls*reshape(imgL,1,[]); % Scaled L-values
    if (isequal(segname,’direct’))
        pm = MeanShift3D(p,h);
    elseif (isequal(segname,’grid’))
        pm = MeanShiftGrid3D(p,h);
    else imgS = img; end
    cN = max(pm(end,:)); ctab = zeros(cN,3);
    for k=1:cN
        i=find(pm(end,:)==k); i2 = H*(pm(1,i)-1)+pm(2,i);
        ctab(k,:) = round(mean([imgR(i2);imgG(i2);imgB(i2)]’,1))
    ;
    end
    imgS = uint8(zeros(H,W,3));
    for k=1:size(pm,2) % Generate a segmentation image
        imgS(pm(2,k),pm(1,k),:) = ctab(pm(end,k),:);
    end
end
```

Given an original image illustrated in the sub-figure(a) of Fig. 6.4 (or ofFig. 6.5), first perform grid sampling (only) based mean shift clustering for image segmentation , the effect of which is demonstrated in Fig. 6.4b. Then perform grid sampling-interpolation based mean shift clustering for image segmentation , the effect of which is demonstrated in Fig. 6.5b. No matter by which method, the original image is indeed segmented in intuitively rather consistent way.

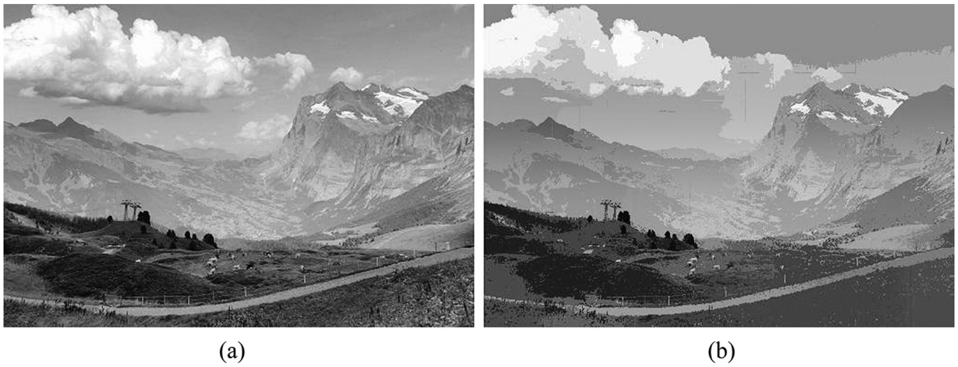

(a) (b)

Fig. 6.4 Grid sampling based mean shift clustering for image segmentation

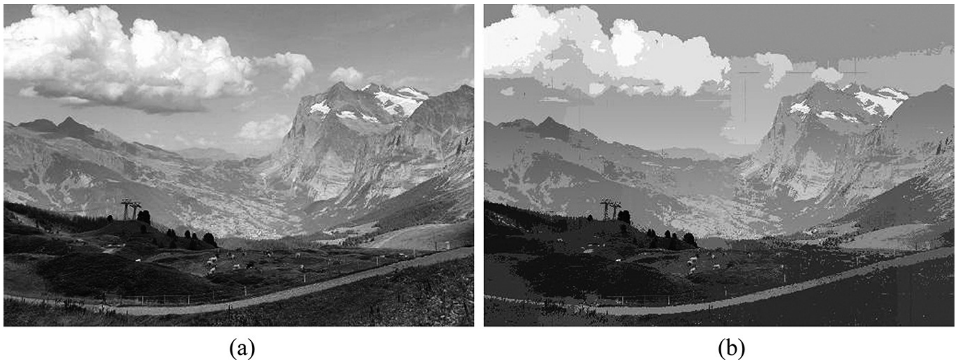

(a) (b)

Fig. 6.5 Grid sampling-interpolation based mean shift clustering for image segmentation

The grid sampling-interpolation based implementation is much more efficient than the grid sampling based implementation (not to say the direct implementation even without taking advantage of grid sampling), yet the former has almost no performance sacrifice compared with the latter—The former and the latter yield almost the same effect of image segmentation —Apply mean shift clustering for image segmentation under various scenarios, the effects of which are demonstrated in Fig. 6.6.

6.1.4 Normalized Cut

The **normalized cut** method [13] is also a representative clustering method, which models relationships among feature points by a graph and performs clustering of them by decomposing the graph into different graphs (sub-graphs) in such way that both inter-graph dissimilarity and intra-graph similarity are maximized, or in other words, in such way that inter-graph similarity is minimized whereas intra-graph similarity is maximized.

More specifically, establish an adjacency matrix $\boldsymbol{W}$ to model relationships among feature points as a *weighted graph* $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}\}$, in which each edge is assigned a weight that somehow reflects similarity between the two vertices connected by the edge. Consider *bipartition*, i.e. two-cluster decomposition

$$[\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}\}] \Rightarrow [\boldsymbol{G}_1 = \{\boldsymbol{V}_1, \boldsymbol{E}_1\}] \cup [\boldsymbol{G}_2 = \{\boldsymbol{V}_2, \boldsymbol{E}_2\}], \tag{6.19}$$

where

$$
\begin{aligned}
\boldsymbol{V} &= \boldsymbol{V}_1 \cup \boldsymbol{V}_2, \quad \boldsymbol{V}_1 \cap \boldsymbol{V}_2 = \emptyset, \\
\boldsymbol{G}_1 &= \boldsymbol{K}(\boldsymbol{V}_1) \cap \boldsymbol{G} = \boldsymbol{G}(\boldsymbol{V}_1), \\
\boldsymbol{G}_2 &= \boldsymbol{K}(\boldsymbol{V}_2) \cap \boldsymbol{G} = \boldsymbol{G}(\boldsymbol{V}_2).
\end{aligned}
$$

In the last two expressions above, $\boldsymbol{K}(\cdot)$ denotes the *complete graph* of the given set of vertices, and $\boldsymbol{G}(\cdot)$ denotes the *induced sub-graph* [14] of the graph by deleting all other vertices not contained in the given set of vertices.

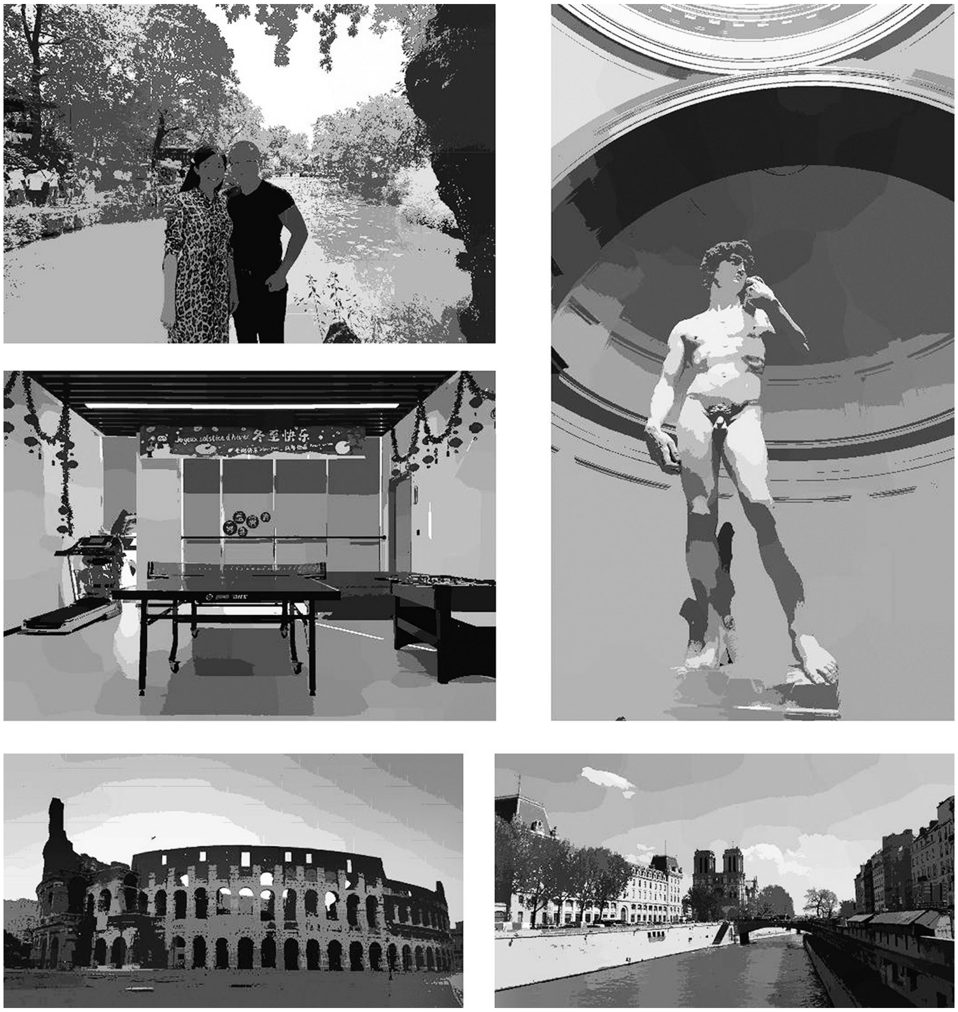


Fig. 6.6 The simple version of mean shift clustering for image segmentation under various scenarios

Normalized cut measure

The normalized cut method consists in finding the optimal bipartition $\boldsymbol{G}_1 \cup \boldsymbol{G}_2$ that minimizes the normalized cut measure [13]

$$
C_{\mathrm{N}}(\boldsymbol{G}_1 \cup \boldsymbol{G}_2) \equiv \frac{C(\boldsymbol{G}_1,\boldsymbol{G}_2)}{C_{\mathrm{A}}(\boldsymbol{G}_1,\boldsymbol{G})} + \frac{C(\boldsymbol{G}_1,\boldsymbol{G}_2)}{C_{\mathrm{A}}(\boldsymbol{G}_2,\boldsymbol{G})}, \tag{6.20}
$$

where [5]

$$
\begin{aligned}
C(\boldsymbol{G}_1,\boldsymbol{G}_2) &\equiv \sum_{v_1\in\boldsymbol{V}_1,v_2\in\boldsymbol{V}_2}\boldsymbol{W}_{v_1,v_2} = \sum_{v_1\in\boldsymbol{V}_1,v_2\in\boldsymbol{V}_2}\boldsymbol{W}_{v_2,v_1},\\
C_{\mathrm{A}}(\boldsymbol{G}_1,\boldsymbol{G}) &\equiv \sum_{v_1\in\boldsymbol{V}_1,v\in\boldsymbol{V}}\boldsymbol{W}_{v_1,v} = \sum_{v_1\in\boldsymbol{V}_1,v\in\boldsymbol{V}}\boldsymbol{W}_{v,v_1},\\
C_{\mathrm{A}}(\boldsymbol{G}_2,\boldsymbol{G}) &\equiv \sum_{v_2\in\boldsymbol{V}_2,v\in\boldsymbol{V}}\boldsymbol{W}_{v_2,v} = \sum_{v_2\in\boldsymbol{V}_2,v\in\boldsymbol{V}}\boldsymbol{W}_{v,v_2}.
\end{aligned}
$$

Set an *indicator vector* $\boldsymbol{x}$ associated with the vertex set

$$\boldsymbol{V} = \{V_1, V_2, \ldots, V_{|\boldsymbol{V}|}\}$$

such that

$$\boldsymbol{x} \equiv \begin{bmatrix} x_1 & x_2 & \cdots & x_{|\boldsymbol{V}|}\end{bmatrix}^{\mathrm{T}} \tag{6.21}$$

takes element values between $\{+1,-1\}$ depending on whether the corresponding vertices belong to $\boldsymbol{G}_1$ or $\boldsymbol{G}_2$ respectively, namely for $i \in \{1,2,\ldots,|\boldsymbol{V}|\}$ one has

$$x_i = \begin{cases} +1, & V_i \in \boldsymbol{V}_1 \\ -1, & V_i \in \boldsymbol{V}_2 \end{cases}$$

Follow definitions of the diagonal matrix of weights $\boldsymbol{D}$ and the Laplacian matrix $\boldsymbol{L} \equiv \boldsymbol{D} - \boldsymbol{W}$ presented in Sect. 3.3.1 in Chap. 3 and have

$$C(\boldsymbol{G}_1,\boldsymbol{G}_2) \equiv \sum_{v_1\in\boldsymbol{V}_1,v_2\in\boldsymbol{V}_2}\boldsymbol{W}_{v_1,v_2} = \left(\frac{\mathbf{1}+\boldsymbol{x}}{2}\right)^{\mathrm{T}}\boldsymbol{W}\left(\frac{\mathbf{1}-\boldsymbol{x}}{2}\right), \tag{6.22a}$$

$$C_{\mathrm{A}}(\boldsymbol{G}_1,\boldsymbol{G}) \equiv \sum_{v_1\in\boldsymbol{V}_1,v\in\boldsymbol{V}}\boldsymbol{W}_{v_1,v} = \left(\frac{\mathbf{1}+\boldsymbol{x}}{2}\right)^{\mathrm{T}}\boldsymbol{W}\mathbf{1}, \tag{6.22b}$$

$$C_{\mathrm{A}}(\boldsymbol{G}_2,\boldsymbol{G}) \equiv \sum_{v_2\in\boldsymbol{V}_2,v\in\boldsymbol{V}}\boldsymbol{W}_{v_2,v} = \left(\frac{\mathbf{1}-\boldsymbol{x}}{2}\right)^{\mathrm{T}}\boldsymbol{W}\mathbf{1}, \tag{6.22c}$$

where $\mathbf{1}$ denotes the all-element-one vector, i.e. $\mathbf{1} \equiv \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^{\mathrm{T}}$.

Substitute (6.22) into (6.20) and obtain

$$\begin{aligned} C_{\mathrm{N}}(\boldsymbol{G}_1 \cup \boldsymbol{G}_2) &\equiv \frac{\left(\frac{\mathbf{1}+\boldsymbol{x}}{2}\right)^{\mathrm{T}} \boldsymbol{W} \left(\frac{\mathbf{1}-\boldsymbol{x}}{2}\right)}{\left(\frac{\mathbf{1}+\boldsymbol{x}}{2}\right)^{\mathrm{T}} \boldsymbol{W}\mathbf{1}} + \frac{\left(\frac{\mathbf{1}+\boldsymbol{x}}{2}\right)^{\mathrm{T}} \boldsymbol{W} \left(\frac{\mathbf{1}-\boldsymbol{x}}{2}\right)}{\left(\frac{\mathbf{1}-\boldsymbol{x}}{2}\right)^{\mathrm{T}} \boldsymbol{W}\mathbf{1}} \\ &= \tfrac{1}{2}\left[\tfrac{\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}+\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}} + \tfrac{\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}}\right] \\ &= \tfrac{(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})}{(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^2-(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})^2} \\ &= \tfrac{(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})}{(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^3-(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})} \end{aligned} \tag{6.23}$$

Note following equalities

$$\mathbf{1}^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{W}) \quad = \mathbf{0} \Longleftrightarrow \mathbf{1}^{\mathrm{T}}\boldsymbol{D} = \mathbf{1}^{\mathrm{T}}\boldsymbol{W}, \tag{6.24a}$$

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x} \quad = \mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}. \tag{6.24b}$$

Substitute (6.24) into the numerator and denominator of (6.23) and obtain respectively

$$\begin{aligned} & (\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}) = (\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}) \\ = \;& (\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^2(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}) = [(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})\boldsymbol{x}]^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{W})[(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})\boldsymbol{x}] \\ & *\,\text{note the first equality of} \qquad (6.24) \\ = \;& [(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})\mathbf{1}]^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{W})[(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})\mathbf{1}] \\ = \;& [(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}]^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{W})[(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}] \end{aligned}$$

and

$$\begin{aligned} & (\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1})^3-(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{W}\mathbf{1}) = (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})^3-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}) \\ & *\,\text{note the second equality of } (6.24) \\ = \;& (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})-2(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})^2(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})+(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}) \\ = \;& [(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}]^{\mathrm{T}}\boldsymbol{D}[(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}]. \end{aligned}$$

So (6.23) becomes

$$C_{\mathrm{N}} \quad (\boldsymbol{G}_1 \cup \boldsymbol{G}_2) = \tfrac{\boldsymbol{y}^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{W})\boldsymbol{y}}{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{y}} = \tfrac{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{y}}, \tag{6.25}$$

where the transformed variable

$$\boldsymbol{y} \equiv (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{x}-(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})\mathbf{1}$$

satisfies the constraint

$$\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{y} = (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x}) - (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{x})(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}) = 0. \tag{6.26}$$

Given $\boldsymbol{y}$ that satisfies the constraint (6.26), the unique $\boldsymbol{x}$ corresponding to $\boldsymbol{y}$ can be determined by solving the linear equation

$$[(\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{I} - \mathbf{1}(\mathbf{1}^{\mathrm{T}}\boldsymbol{D})]\boldsymbol{x} = \boldsymbol{y} \tag{6.27}$$

namely expressed in details as (each $d_i > 0$ for $i \in \{1, \dots, |\boldsymbol{V}|\}$)

$$\begin{bmatrix} \sum\limits_{i\neq 1} d_i & -d_2 & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\ -d_1 & \sum\limits_{i\neq 2} d_i & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\ -d_1 & -d_2 & \sum\limits_{i\neq 3} d_i & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -d_1 & -d_2 & -d_3 & \cdots & \sum\limits_{i\neq |\boldsymbol{V}|-1} d_i & -d_{|\boldsymbol{V}|} \\ -d_1 & -d_2 & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & \sum\limits_{i\neq |\boldsymbol{V}|} d_i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{|\boldsymbol{V}|-1} \\ x_{|\boldsymbol{V}|} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{|\boldsymbol{V}|-1} \\ y_{|\boldsymbol{V}|} \end{bmatrix}.$$

For the $|\boldsymbol{V}|$-by-$|\boldsymbol{V}|$ square matrix $\boldsymbol{L_D} \equiv (\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1})\boldsymbol{I} - \mathbf{1}(\mathbf{1}^{\mathrm{T}}\boldsymbol{D})$ on the left side of (6.27), one has

$$\begin{aligned} \boldsymbol{z}^{\mathrm{T}}\boldsymbol{L_D}\boldsymbol{z} &= \left(\sum_{i=1}^{|\boldsymbol{V}|} d_i\right)\left(\sum_{i=1}^{|\boldsymbol{V}|} z_i^2\right) - \left(\sum_{i=1}^{|\boldsymbol{V}|} z_i\right)\left(\sum_{i=1}^{|\boldsymbol{V}|} d_i z_i\right) \quad \left(\forall \boldsymbol{z} = \begin{bmatrix} z_1 & \cdots & z_{|\boldsymbol{V}|}\end{bmatrix}^{\mathrm{T}}\right) \\ &= \sum_{1\leqslant i<j\leqslant|\boldsymbol{V}|} \left(\sqrt{d_i}z_j - \sqrt{d_j}z_i\right)^2 = \sum_{1\leqslant i<j\leqslant|\boldsymbol{V}|} d_i d_j \left(\frac{z_j}{\sqrt{d_j}} - \frac{z_i}{\sqrt{d_i}}\right)^2 \geqslant 0. \end{aligned} \tag{6.28}$$

The inequality in (6.28) implies that $\boldsymbol{L_D}$ is positive semi-definite. The equality in (6.28) holds if and only if

$$\tfrac{z_1}{\sqrt{d_1}} = \tfrac{z_2}{\sqrt{d_2}} = \cdots = \tfrac{z_{|\boldsymbol{V}|}}{\sqrt{d_{|\boldsymbol{V}|}}} \Longleftrightarrow \boldsymbol{z} \propto \begin{bmatrix} \sqrt{d_1} & \sqrt{d_2} & \cdots & \sqrt{d_{|\boldsymbol{V}|}} \end{bmatrix}^{\mathrm{T}}. \tag{6.29}$$

So $\boldsymbol{L_D}$ has one but only one eigenvector associated with the eigenvalue 0, as specified in (6.29), and has a rank of $|\boldsymbol{V}| - 1$.

The fact that $\boldsymbol{L_D}$ is positive semi-definite while having a rank of $|\boldsymbol{V}| - 1$ implies that the linear equation (6.27) has exactly one redundant variable.[6] In practice, one may just choose x_1 as the redundant indicator variable and set $x_1 = +1$ namely always assign V_1 to $\boldsymbol{G}_1$. Then the other $|\boldsymbol{V}| - 1$ indicator variables x_2, x_3, ..., $x_{|\boldsymbol{V}|}$ can be determined by solving the linear equation (refer to Sect. F.1 in Appendix F)

$$
\begin{bmatrix}
-d_2 & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\
\sum\limits_{i\neq 2} d_i & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\
-d_2 & \sum\limits_{i\neq 3} d_i & \cdots & -d_{|\boldsymbol{V}|-1} & -d_{|\boldsymbol{V}|} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
-d_2 & -d_3 & \cdots & \sum\limits_{i\neq |\boldsymbol{V}|-1} d_i & -d_{|\boldsymbol{V}|} \\
-d_2 & -d_3 & \cdots & -d_{|\boldsymbol{V}|-1} & \sum\limits_{i\neq |\boldsymbol{V}|} d_i
\end{bmatrix}
\begin{bmatrix}
x_2 \\ x_3 \\ \vdots \\ x_{|\boldsymbol{V}|-1} \\ x_{|\boldsymbol{V}|}
\end{bmatrix}
=
\begin{bmatrix}
y_1 - \sum\limits_{i\neq 1} d_i \\ y_2 + d_1 \\ y_3 + d_1 \\ \vdots \\ y_{|\boldsymbol{V}|-1} + d_1 \\ y_{|\boldsymbol{V}|} + d_1
\end{bmatrix}. \tag{6.30}
$$

Normalized cut optimization solving

Following previous analysis, the core of the normalized cut method consists in solving the constrained optimization problem

$$
\begin{aligned}
\boldsymbol{y} = \quad & \operatorname{argmin}_{\boldsymbol{y}} \frac{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{y}} = \sqrt{\boldsymbol{D}^{-1}}\operatorname{argmin}_{\bar{\boldsymbol{y}}} \frac{\bar{\boldsymbol{y}}^{\mathrm{T}}\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}\bar{\boldsymbol{y}}}{\bar{\boldsymbol{y}}^{\mathrm{T}}\bar{\boldsymbol{y}}} \\
& \mathrm{s.t.}\, \mathbf{1}^{\mathrm{T}}\mathbf{D}\mathbf{y} = \mathbf{0} \Longleftrightarrow \mathbf{1}^{\mathrm{T}}\sqrt{\mathbf{D}}\bar{\mathbf{y}} \qquad (\mathbf{y} = \sqrt{\mathbf{D}^{-1}}\bar{\mathbf{y}})
\end{aligned} \tag{6.31}
$$

In the context of graph bipartition, strictly speaking, (6.31) is a *discrete optimization* problem, which is indeed difficult. For tractability in practice, one may just relax the discrete version of the optimization problem (6.31) to its continuous counterpart, and such relaxation still demonstrates acceptable performance in practical applications.

As presented in Sect. 3.3.1 in Chap. 3, the Laplacian matrix $\boldsymbol{L}$ is positive semi-definite, has one but only one eigenvector associated with the eigenvalue 0, and has a rank of $|\boldsymbol{V}| - 1$. The matrix *congruent transformation* from $\boldsymbol{L}$ to $\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}$ does not have any influence on these properties, so the transformed Laplacian matrix $\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}$ is also positive semi-definite, has one but only one eigenvector associated with the eigenvalue 0, and has a rank of $|\boldsymbol{V}| - 1$.

Since the transformed Laplacian matrix $\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}$ is real symmetric and positive semi-definite, it can be decomposed into [15]

$$
\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}} = [\boldsymbol{U}_1 \quad \boldsymbol{U}_2 \quad \cdots \quad \boldsymbol{U}_{|\boldsymbol{V}|}]
\begin{bmatrix}
\sigma_1 & & & \\
& \sigma_2 & & \\
& & \ddots & \\
& & & \sigma_{|\boldsymbol{V}|}
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{U}_1^{\mathrm{T}} \\ \boldsymbol{U}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{U}_{|\boldsymbol{V}|}^{\mathrm{T}}
\end{bmatrix}
\equiv \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^{\mathrm{T}}, \tag{6.32}
$$

where

$$
\sigma_1 = 0 < \sigma_2 \leqslant \sigma_3 \leqslant \cdots \leqslant \sigma_{|\boldsymbol{V}|}
$$

denote the eigenvalues of $\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}$, and $\boldsymbol{U}$ denotes the orthonormal matrix consisting of the normalized eigenvectors associated with $\sigma_1 = 0$, σ_2, ..., $\sigma_{|\boldsymbol{V}|}$ respectively.

For the Laplacian matrix $\boldsymbol{L}$, the eigenvector associated with the eigenvalue 0 is the all-element-one vector$\boldsymbol{1}$. So for the transformed Laplacian matrix $\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}$, the normalized eigenvector $\boldsymbol{U}_1$ associated with the eigenvalue $\sigma_1 = 0$ follows

$$\boldsymbol{U}_1 \propto \sqrt{\boldsymbol{D}}\boldsymbol{1}$$

and hence the constraint (6.26) can be formalized equivalently as

$$\boldsymbol{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{y} = 0 \Longleftrightarrow \left(\sqrt{\boldsymbol{D}}\boldsymbol{1}\right)^{\mathrm{T}}\left(\sqrt{\boldsymbol{D}}\boldsymbol{y}\right) = 0 \Longleftrightarrow \boldsymbol{U}_1^{\mathrm{T}}\bar{\boldsymbol{y}} = 0. \tag{6.33}$$

So $\bar{\boldsymbol{y}}$ must be a linear combination of $\boldsymbol{U}_2$, $\boldsymbol{U}_3$, ..., $\boldsymbol{U}_{|\boldsymbol{V}|}$. Denote $\bar{\boldsymbol{y}}$ generically as

$$\bar{\boldsymbol{y}} = c_2\boldsymbol{U}_2 + c_3\boldsymbol{U}_3 + \cdots + c_{|\boldsymbol{V}|}\boldsymbol{U}_{|\boldsymbol{V}|}$$

and substitute it into the objective function described in (6.31)

$$\begin{aligned} \frac{\bar{\boldsymbol{y}}^{\mathrm{T}}\sqrt{\boldsymbol{D}^{-1}}\boldsymbol{L}\sqrt{\boldsymbol{D}^{-1}}\bar{\boldsymbol{y}}}{\bar{\boldsymbol{y}}^{\mathrm{T}}\bar{\boldsymbol{y}}} &= \frac{\begin{bmatrix} 0 & c_2 & \cdots & c_{|\boldsymbol{V}|} \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_{|\boldsymbol{V}|} \end{bmatrix} \begin{bmatrix} 0 \\ c_2 \\ \vdots \\ c_{|\boldsymbol{V}|} \end{bmatrix}}{c_2^2 + c_3^2 + \cdots + c_{|\boldsymbol{V}|}^2} \\ &= \frac{c_2^2\sigma_2 + c_3^2\sigma_3 + \cdots + c_{|\boldsymbol{V}|}^2\sigma_{|\boldsymbol{V}|}}{c_2^2 + c_3^2 + \cdots + c_{|\boldsymbol{V}|}^2} \geqslant \sigma_2, \end{aligned} \tag{6.34}$$

where the equality holds if

$$c_2 \neq 0, \quad c_3 = 0, \quad \cdots \quad , \quad c_{|\boldsymbol{V}|} = 0 \Longleftrightarrow \bar{\boldsymbol{y}} \propto \boldsymbol{U}_2. \tag{6.35}$$

Once the optimal solution of $\bar{\boldsymbol{y}}$ given in (6.35) is obtained, the optimal solution of $\boldsymbol{y} = \sqrt{\boldsymbol{D}^{-1}}\bar{\boldsymbol{y}}$ can be obtained as well, and the optimal solution of $\boldsymbol{x}$ can be further solved via (6.27) and (6.30). **Thresholding of the optimal solution**

As clarified previously, the discrete version of the optimization problem (6.31) is usually relaxed to its continuous counterpart in practical applications. So naturally, the obtained optimal solution of $\boldsymbol{x}$ takes continuous values instead of the discrete values $\{+1, -1\}$. For final output of graph bipartition, one needs to perform thresholding of the optimal solution.

There are a number of possibilities for such thresholding. The simplest way would be to fix a threshold such as 0, and to perform thresholding of the optimal solution as

$$x_i := \begin{cases} +1, & x_i > 0 \\ -1, & x_i < 0 \end{cases}$$

where $:=$ denotes *value assignment* rather than equality.

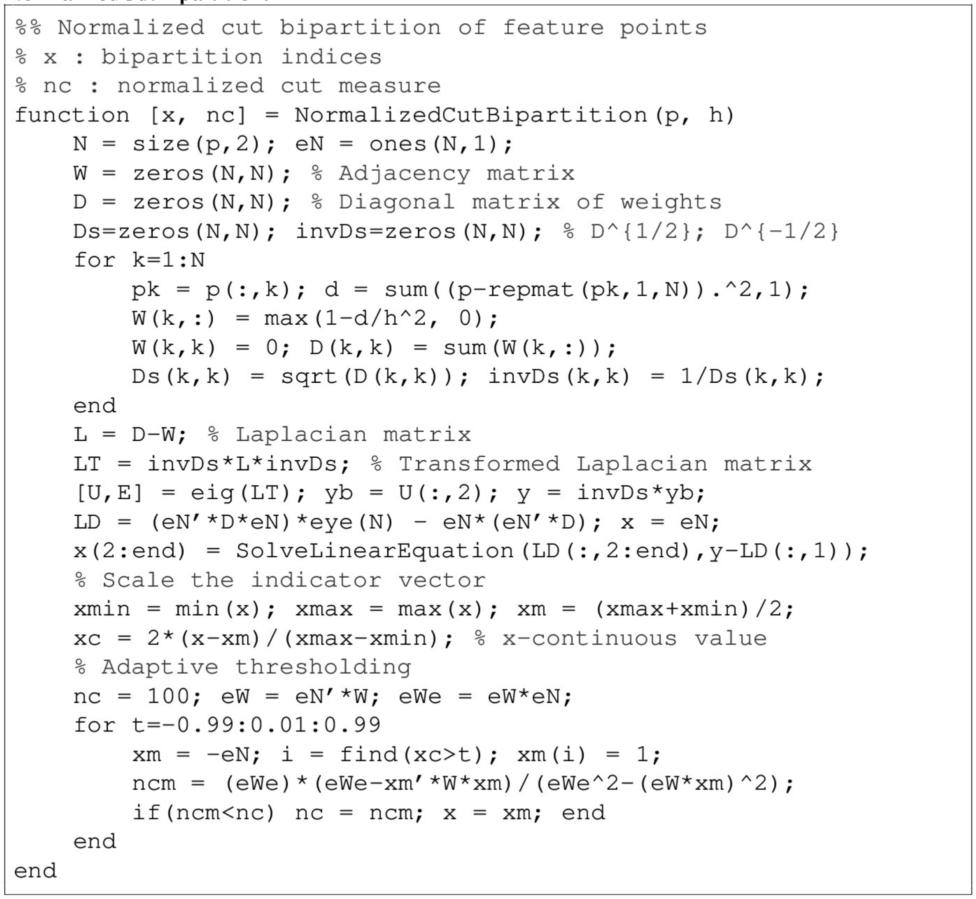
Since the cost of computing the normalized cut measure given a specified indicator vector $\boldsymbol{x}$ is moderate, a more reasonable way for thresholding of the optimal solution is to perform *adaptive thresholding*, namely to scale the indicator vector values to a normalized range, try various thresholds, and then select the best threshold namely the one corresponding to the minimal normalized cut measure.

Recursive clustering

For clustering, the normalized cut method is realized in recursive way, namely to perform the optimal (or semi-optimal) bipartition of sub-graphs recursively until normalized cut measures incurred by further bipartition exceed certain limit.

Matlab code for normalized cut bipartition of feature points is given as follows. The triangular kernel, which is already involved in presentation of the mean shift method, is adopted to evaluate weights of similarity among feature points. The parameter *h* denotes the triangular kernel bandwidth .

NormalizedCutBipartition.m

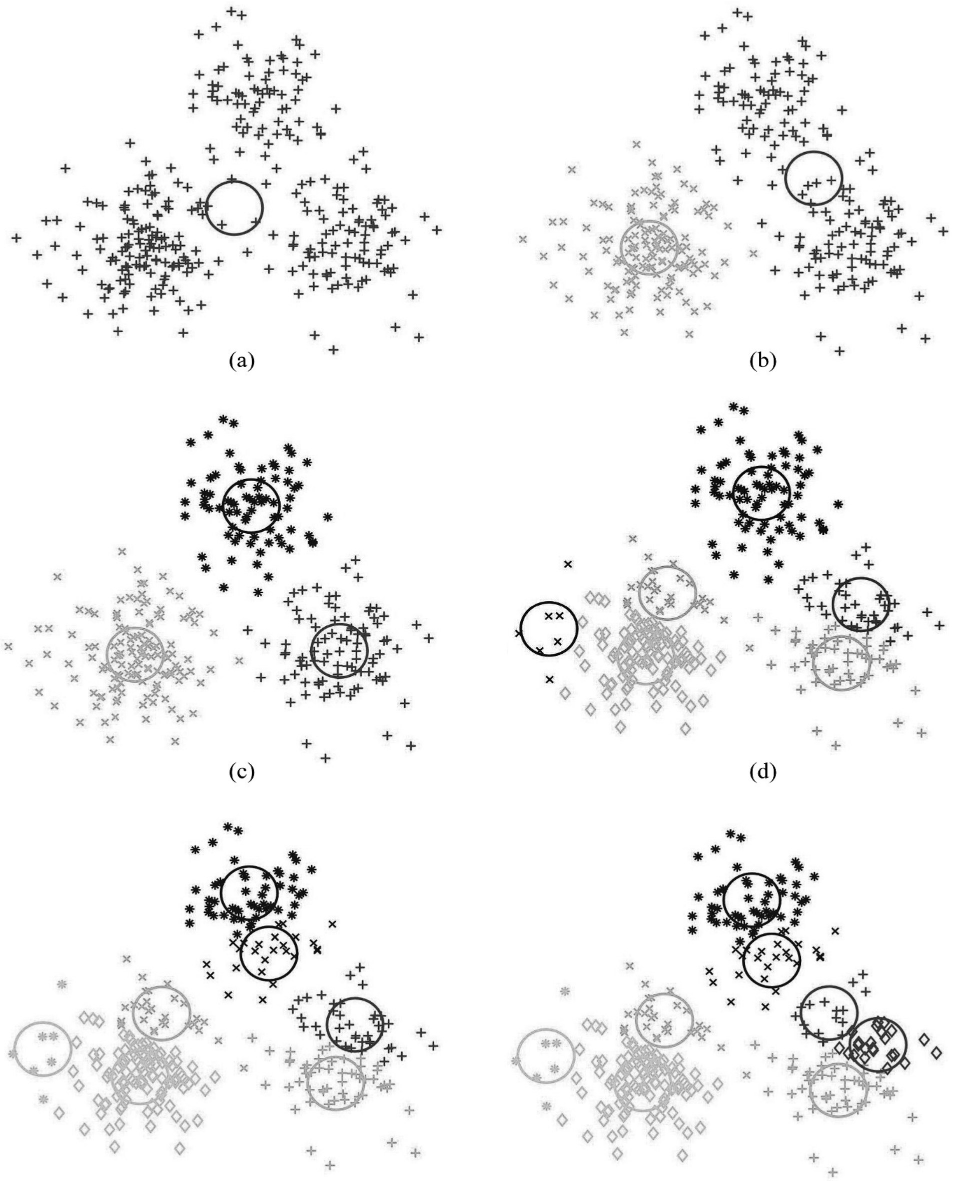


```
%% Normalized cut bipartition of feature points
% x : bipartition indices
% nc : normalized cut measure
function [x, nc] = NormalizedCutBipartition(p, h)
    N = size(p,2); eN = ones(N,1);
    W = zeros(N,N); % Adjacency matrix
    D = zeros(N,N); % Diagonal matrix of weights
    Ds=zeros(N,N); invDs=zeros(N,N); % D^{1/2}; D^{-1/2}
    for k=1:N
        pk = p(:,k); d = sum((p-repmat(pk,1,N)).^2,1);
        W(k,:) = max(1-d/h^2, 0);
        W(k,k) = 0; D(k,k) = sum(W(k,:));
        Ds(k,k) = sqrt(D(k,k)); invDs(k,k) = 1/Ds(k,k);
    end
    L = D-W; % Laplacian matrix
    LT = invDs*L*invDs; % Transformed Laplacian matrix
    [U,E] = eig(LT); yb = U(:,2); y = invDs*yb;
    LD = (eN’*D*eN)*eye(N) - eN*(eN’*D); x = eN;
    x(2:end) = SolveLinearEquation(LD(:,2:end),y-LD(:,1));
    % Scale the indicator vector
    xmin = min(x); xmax = max(x); xm = (xmax+xmin)/2;
    xc = 2*(x-xm)/(xmax-xmin); % x-continuous value
    % Adaptive thresholding
    nc = 100; eW = eN’*W; eWe = eW*eN;
    for t=-0.99:0.01:0.99
        xm = -eN; i = find(xc>t); xm(i) = 1;
        ncm = (eWe)*(eWe-xm’*W*xm)/(eWe^2-(eW*xm)^2);
        if(ncm<nc) nc = ncm; x = xm; end
    end
end
```

Matlab code for normalized cut clustering of feature points is given as follows. The parameter *ncT* denotes the normalized cut measure limit. If the optimal bipartition returned by **NormalizedCutBipartition.m** results in a normalized cut measure lower than the limit, then the bipartition is accepted and normalized cut clustering is performed recursively further for the two sub-graphs. Otherwise, the bipartition is not accepted and normalized cut clustering is stopped for current branch of recursive execution.

NormalizedCut.m

```
%% Normalized cut clustering of feature points
% pc : [points; clustering indices]
function pc = NormalizedCut(p, h, ncT)
    [x, nc] = NormalizedCutBipartition(p, h);
    % No bipartition if the n.c. measure is large
    if (nc>ncT) pc = [p;ones(1,size(p,2))]; return; end
    i1 = find(x==1); i2 = find(x==-1);
    pc1 = NormalizedCut(p(:,i1), h, ncT); % Recursion
    pc2 = NormalizedCut(p(:,i2), h, ncT); % Recursion
    pc =[p;zeros(1,size(p,2))];
    pc(end,i1) = 2*pc1(end,:)+1; % Recursive indexing
    pc(end,i2) = 2*pc2(end,:)-1; % Recursive indexing
end
```



(a) (b)

(c) (d)

(e) (f)

Fig. 6.7 Normalized cut clustering with various limits ncT **a** $ncT = 0.01$; **b** $ncT = 0.02 \sim 0.03$; **c** $ncT = 0.04 \sim 0.64$; **d** $ncT = 0.65$; **e** $ncT = 0.66 \sim 0.68$; (f) $ncT = 0.69 \sim 0.70$

Still take the example of feature points illustrated in Fig. 6.2a. Apply the normalized cut method to perform clustering of the feature points, effects of which are demonstrated in Fig. 6.7. Various normalized cut measure limits *ncT* (more specifically, in the range from 0.01 to 1.00 at the step interval of 0.01) are tried respectively.

Figure 6.7a demonstrates the effect of normalized cut clustering with the normalized cut measure limit *ncT* being set to 0.01, where a holistic cluster of feature points is finally obtained after recursive graph bipartition. Since the normalized cut measure limit is too small, no bipartition is accepted at all. Figure 6.7b demonstrates the effect of normalized cut clustering with the normalized cut measure limit *ncT* being set in the range from 0.02 to 0.03, where two clusters of feature points are finally obtained after recursive graph bipartition. As the normalized cut measure limit *ncT* increases, bipartition of the original graph begins to be accepted. Figure 6.7c demonstrates the effect of normalized cut clustering with the normalized cut measure limit *ncT* being set in the range from 0.04 to 0.64, where three clusters of feature points are finallyobtained after recursive graph bipartition. Figure 6.7d–f demonstrate the effects of normalized cut clustering with the normalized cut measure limit *ncT* being set to 0.65, being set in the range from 0.66 to 0.68, and being set in the range from 0.69 to 0.70 respectively, where six clusters of feature points, seven clusters of feature points, and eight clusters of feature points are obtained after recursive graph bipartition respectively. In each sub-figure, the clusters are distinguished by marks of different shapes and of different colors.

It is worth noting that the effect of normalized cut clustering demonstrated in Fig. 6.7c, i.e. the effect of generating three clusters of feature points, is absolutely most stable in the sense that such effect of generating three clusters of feature points is commonly obtained for the absolutely largest range of the normalized cut measure limit *ncT*. This is rather consistent with the intuition that the feature points seem to indeed form three clusters. Motivated by such consistency, one may naturally adopt an adaptive mechanism for tuning of the normalized cut measure limit *ncT*, namely to select the largest range of *ncT* that results in the same clustering effect or in other words to select the most stable clustering effect.

6.1.5 Segmentation

A special form of clustering image pixels is to segment the image into distinct regions. As already mentioned in Sect. 2.3 in Chap. 2, humans' biological vision systems possess a mechanism that strengthens brightness or intensity contrast (refer to Appendix B), so humans' vision is rather sensitive to object edges which tend to cause considerable brightness contrast. Object edges are geometric entities that naturally segment the scenario appearance into distinct regions, so in early years, researches on edge analysis based image segmentation are directly inspired by such daily-life experiences, both for conventional (two-dimensional) images [16] and for range (three-dimensional) images [17].

Image segmentation can also be realized using generic clustering methods. For example, effects of mean shift clustering for image segmentation have already been demonstrated in Figs. 6.4, 6.5, and 6.6 in Sect. 6.1.3. The normalized cut method has also been strongly promoting research works on image segmentation [10, 18–21].

In the last decade, supported by revolutionary progress in hardware, the renaissance of deep learning has revolutionized a wide range of research areas. A brief review of deep learning methods has already been given in Sect. 2.8 in Chap. 2.

Prosperity of deep learning methods has also considerably benefited the research area of image segmentation (including semantic segmentation and instance segmentation) and has largely enhanced segmentation performance in practical applications. Readers may refer to [22] for a survey of deep learning based segmentation methods.

6.2 Known Feature-Model Relationship Oriented Model Fitting

In practical applications, one may need to organize features points by fitting them with certain parametric model, be the feature-model relationship known or unknown. A parametric model may refer to common shapes, such as lines, circles, triangles, rectangles etc., which are easy for conception on one hand and convenient for further processing on the other hand. A parametric model may also refer to abstract geometric structures. **Model fitting** aims at finding a set of model parameters that enable the parametric model to fit feature points in the best way or at least in a good way.

Here, the feature-model relationship being known means whether each feature point is or is not associated with the parametric model is known. In contrast, the feature-model relationship being unknown means whether each feature point is or is not associated with the parametric model is unknown. In this sub-section the author focuses on known feature-model relationship oriented model fitting, whereas in next sub-section the author focuses on unknown feature-model relationship oriented model fitting.

6.2.1 Least Mean Squares (LMS) Method

The **least mean squares** (LMS) method is a natural and commonly-adopted instantiation of a general spirit namely the spirit of **maximum likelihood**. Given a set of independent data pairs denoted as

$$\{\boldsymbol{x}_{\mathrm{I},1}, \boldsymbol{y}_{\mathrm{O},1}\}, \quad \{\boldsymbol{x}_{\mathrm{I},2}, \boldsymbol{y}_{\mathrm{O},2}\}, \quad \dots \quad , \quad \{\boldsymbol{x}_{\mathrm{I},N-1}, \boldsymbol{y}_{\mathrm{O},N-1}\}, \quad \{\boldsymbol{x}_{\mathrm{I},N}, \boldsymbol{y}_{\mathrm{O},N}\}.$$

From the probabilistic perspective, the relationship between the input $\boldsymbol{x}_{\mathrm{I}}$ and the output $\boldsymbol{y}_{\mathrm{O}}$ of a generic data pair is described by a probabilistic model

$$\boldsymbol{y}_{\mathrm{O}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I}}) + \boldsymbol{\gamma}. \tag{6.36}$$

The parametric function $\boldsymbol{f}(\boldsymbol{x}, \cdot)$ in (6.36) denotes a target model of known structure but of unknown (to be identified) parameter set $\boldsymbol{x}$. The variable $\boldsymbol{\gamma}$ in (6.36) denotes probabilistic uncertainty that can be random errors or noises, modelling errors, etc.

In the spirit of maximum likelihood, model fitting, i.e. identifying the unknown parameters $\boldsymbol{x}$ is to search for $\boldsymbol{x}$ that maximizes the joint likelihood

$$p(\boldsymbol{y}_{\mathrm{O},1}|\boldsymbol{x}_{\mathrm{I},1})p(\boldsymbol{y}_{\mathrm{O},2}|\boldsymbol{x}_{\mathrm{I},2})\dots p(\boldsymbol{y}_{\mathrm{O},N-1}|\boldsymbol{x}_{\mathrm{I},N-1})p(\boldsymbol{y}_{\mathrm{O},N}|\boldsymbol{x}_{\mathrm{I},N}).$$

In other words, model fitting is formalized as the optimization problem

$$\boldsymbol{x} = \mathrm{argmax}_{\boldsymbol{x}} \prod_{i=1}^{N} p(\boldsymbol{y}_{\mathrm{O},i}|\boldsymbol{x}_{\mathrm{I},i}) \Longleftrightarrow \boldsymbol{x} = \mathrm{argmax}_{\boldsymbol{x}} \prod_{i=1}^{N} p(\boldsymbol{\gamma}_i). \tag{6.37}$$

As a convention, suppose that the model uncertainty $\boldsymbol{\gamma}$ follows a normal distribution, i.e. a Gaussian distribution $\boldsymbol{\gamma} \sim N(0, \boldsymbol{\Sigma})$. Then the general formalism (6.37) becomes the concrete formalism

$$\begin{aligned}
\boldsymbol{x} =& \operatorname{argmax}_{\boldsymbol{x}} \prod_{i=1}^{N} \frac{1}{\sqrt{(2\pi)^m |\boldsymbol{\Sigma}|}} \mathrm{e}^{-\frac{1}{2}(\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i}))^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i}))} \\
\Leftrightarrow \quad \boldsymbol{x} =& \operatorname*{argmin}_{\boldsymbol{x}} \sum_{i=1}^{N} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i}))^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})) \\
\Leftrightarrow \quad \boldsymbol{x} =& \operatorname*{argmin}_{\boldsymbol{x}} \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i}))^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})).
\end{aligned} \tag{6.38}$$

If the m dimensions of the output $\boldsymbol{y}_{\mathrm{O}}$ metric are mutually independent and are consistently scaled with each other, then the covariance $\boldsymbol{\Sigma}$ is reduced to the identity matrix, i.e. $\boldsymbol{\Sigma} = \boldsymbol{I}$ and (6.38) is reduced accordingly to

$$\begin{aligned}
\boldsymbol{x} &= \operatorname*{argmin}_{\boldsymbol{x}} \sum_{i=1}^{N} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i}))^{\mathrm{T}} (\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})) \\
&= \operatorname*{argmin}_{\boldsymbol{x}} \sum_{i=1}^{N} \|\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})\|_2^2 = \operatorname*{argmin}_{\boldsymbol{x}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})\|_2^2
\end{aligned} \tag{6.39}$$

which formalizes the **least mean squares** (LMS) method.

Strictly speaking, the method that relies on solving the optimization problem (6.39) without the factor $1/N$ should be called the **least squares** method without the term “mean”. In fact, for optimization involving the error squares themselves, the least mean squares method and the least squares method are equivalent, because they differ only by a constant factor which has no influence on the optimization solution. In contrast, for optimization involving not only the error squares themselves but also extra terms such as regularization terms of parameter norms, the least mean squares method and the least squares method do make a difference. Normally in such case, the least mean squares method is preferred, because the optimization solution obtained via the least mean squares method is not susceptible to N namely the amount of data. Otherwise, if the least squares method is adopted, increasing of N tends to incline the optimization solution towards minimization of error squares and towards negligence of extra terms that normally reflect certain important concerns as well.[7]

The least mean squares method and the least squares method are not distinguished from each other and are commonly referred to as the least mean squares method in relevant presentations throughout this book, only if confusion is not incurred.

Linear fitting

Suppose the parametric target model $\boldsymbol{f}(\boldsymbol{x}, \cdot)$ in (6.36) is linear

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I}}) = \boldsymbol{a}^{\mathrm{T}} \boldsymbol{x}_{\mathrm{I}} + b = \begin{bmatrix} \boldsymbol{a} & b \end{bmatrix}^{\mathrm{T}} \boldsymbol{x}_{\mathrm{I}}^{A}, \tag{6.40}$$

where the linear coefficients $\boldsymbol{a}$ and b form the parameter set $\boldsymbol{x}$ and $\boldsymbol{x}_{\mathrm{I}}^{A}$ denotes the augmented vector of $\boldsymbol{x}_{\mathrm{I}}$, namely

$$\boldsymbol{x} \equiv \begin{bmatrix} \boldsymbol{a} \\ b \end{bmatrix}, \quad \boldsymbol{x}_{\mathrm{I}}^{A} \equiv \begin{bmatrix} \boldsymbol{x}_{\mathrm{I}} \\ 1 \end{bmatrix}.$$

The optimization problem (6.39) is reduced to

$$\boldsymbol{x} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| y_{\mathrm{O},i} - \boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}_{\mathrm{I},i}^{A} \right\|_2^2, \tag{6.41}$$

As presented in Sect. F.1 in Appendix F, the linear optimization problem version (6.41) can be solved by solving the linear equation

$$\begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},2}^{\mathrm{T}} & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},N}^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} y_{\mathrm{O},1} \\ y_{\mathrm{O},2} \\ \vdots \\ y_{\mathrm{O},N-1} \\ y_{\mathrm{O},N} \end{bmatrix}$$

which has the solution specified in (F.7).

An example of linear fitting is *line fitting* on the plane. Given a set of planar points

$$(x_1, y_1), \quad (x_2, y_2), \quad \ldots \quad , \quad (x_{N-1}, y_{N-1}), \quad (x_N, y_N).$$

A way of parametrizing the target model for line fitting is

$$y_i = k x_i + t \quad (i \in \{1, \ldots, N\}). \tag{6.42}$$

The parameter set $\{k, t\}$ in (6.42) is recovered by solving the linear equation

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_{N-1} & 1 \\ x_N & 1 \end{bmatrix} \begin{bmatrix} k \\ t \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}. \tag{6.43}$$

Matlab code for line fitting via (6.43), i.e. **LineFitting2D_I** is given as follows.

Line fitting via (6.43) is *ill-conditioned* when the slope k is large. The problem becomes more and more serious as the slope k becomes larger and larger, and finally becomes the *singularity* problem when the slope k becomes infinite (namely the given planar points are located on a vertical line). To overcome the problem, a better way of parametrizing the target model for line fitting is

$$ax_i + by_i + c = 0 \quad (i \in \{1, \ldots, N\}). \tag{6.44}$$

The parameter set $\{a, b, c\}$ in (6.44) is recovered by solving the homogeneous linear equation (see Sect. F.2 in Appendix F)

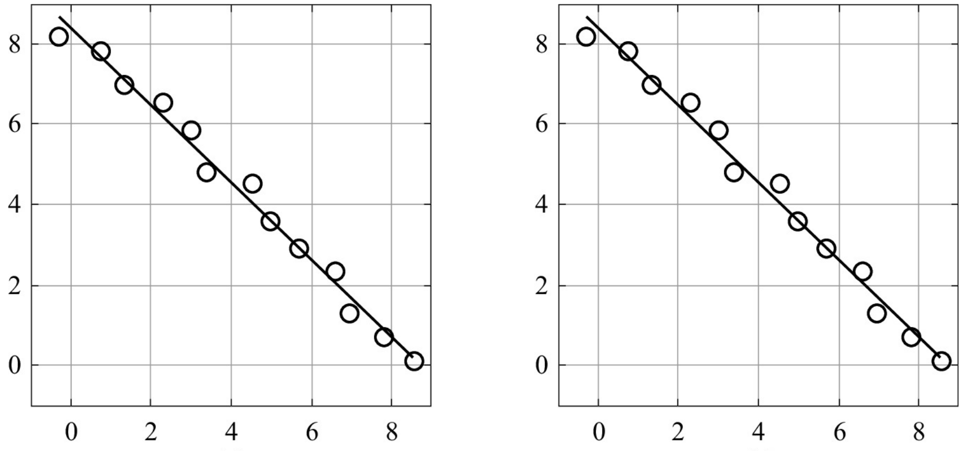
$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_{N-1} & y_{N-1} & 1 \\ x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \mathbf{0}. \tag{6.45}$$

Matlab code for line fitting via (6.45), i.e. **LineFitting2D_II** is given as follows.

LineFitting.m

```
%% Line fitting
function kt = LineFitting2D_I(x,y) % y = k x + b
    x = reshape(x,[],1); y = reshape(y,[],1);
    kt = SolveLinearEquation([x, ones(size(x))], y);
end
function abc = LineFitting2D_II(x,y) % a x + b y + c = 0
    x = reshape(x,[],1); y = reshape(y,[],1);
    A = [x, y, ones(size(x))];
    abc = SolveHomogeneousLinearEquation(A);
end
```

An example of planar points are indicated by circles illustrated in Fig. 6.8. The effect of line fitting via (6.43) and that via (6.45) are demonstrated in Fig. 6.8a, b respectively. The two kinds of line fitting seem to have no intuitive difference, whereas nuances exist in the two sets of parameters associated with them, as given in Table 6.1.



(a)

(b)

Fig. 6.8 Line fitting **a** target model $y = kx + t$; **b** target model $ax + by + c = 0$

Nonlinear fitting

Suppose the parametric target model $\boldsymbol{f}(\boldsymbol{x}, \cdot)$ in (6.36) is nonlinear, one may resort to the method presented in Sect. F.3 in Appendix F to solve the problem of nonlinear fitting.

Table 6.1 Parameters of line fitting

$\{\boldsymbol{k}, \boldsymbol{-1}, \boldsymbol{t}\}$ **normalized**	$-0.113\,149$	$-0.117\,489$	**0.986 607**
$\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$ **normalized**	$-0.113\,400$	$-0.116\,950$	0.986 642

An example of nonlinear fitting is *circle fitting* on the plane. Given a set of planar points

$$(x_1, y_1), \quad (x_2, y_2), \quad \dots \quad , \quad (x_{N-1}, y_{N-1}), \quad (x_N, y_N)$$

that represent partially the contour of a circle.[8] A way of parametrizing the target model for circle fitting is

$$0 = \sqrt{(x_i - x)^2 + (y_i - y)^2} - r \quad (i \in \{1, \dots, N\}), \tag{6.46}$$

where (*x*, *y*) denotes the circle center and *r* denotes the circle radius. The parameter set $\{x, y, r\}$ in (6.46) is recovered by solving the nonlinear equation

$$\begin{bmatrix} \sqrt{(x_1 - x)^2 + (y_1 - y)^2} \\ \sqrt{(x_2 - x)^2 + (y_2 - y)^2} \\ \vdots \\ \sqrt{(x_{N-1} - x)^2 + (y_{N-1} - y)^2} \\ \sqrt{(x_N - x)^2 + (y_N - y)^2} \end{bmatrix} - r = \mathbf{0}. \tag{6.47}$$

To solve (6.47), a proper initial estimate of the parameter set $\{x, y, r\}$ is needed, which can be obtained by fitting a circle to three planar points among the given ones. Matlab code for circle fitting is given as follows.

CircleFitting.m

```
%% Circle fitting
function xyr = CircleFitting(x,y) % (x-x0)^2+(y-y0)^2=r^2
    p=[reshape(x,1,[]);reshape(y,1,[])]; pf=zeros(length(x),1);
    % Initial estimate of the center and radius
    xyr = FitThreePoints([x(1);y(1)],[x(end);y(end)], ...
        [x(round((1+end)/2));y(round((1+end)/2))]);
    fprintf(’Circle initial [x0, y0, r]: %f, %f, %f=>\n’, xyr);
    res = CF(xyr,p); fprintf(’Residual initial: %f\n’,res’*res);
    % Refined estimate of the center and radius
    for k = 1:5
        xyr=SolveParametricNonlinearEquation(@CF,xyr,p,pf,1);
        fprintf(’Circle [x0, y0, r]: %f, %f, %f=>\n’, xyr);
        res = CF(xyr,p); res = res’*res;
        fprintf(’Residual after %d-th refinement: %f\n’,k,res);
    end
end
% Fit a circle that passes three given points
function xyr = FitThreePoints(a, b, c)
    x1 = (a(1)+b(1))/2; y1 = (a(2)+b(2))/2;
    dx1 = b(1)-a(1); dx2 = c(1)-a(1); dx3 = c(1)-b(1);
    dy1 = b(2)-a(2); dy2 = c(2)-a(2); dy3 = c(2)-b(2);
    xy = [x1;y1] + [1,0]*...
        ([dy1,-dy3;-dx1,dx3]\[
\bUnALT{}dx2/2;dy2/2])*[dy1;-dx1];
    xyr = [xy;sqrt((a-xy)’*(a-xy))];
end
% Target model function
function res = CF(xyr, p)
    n = size(p,2); res = zeros(n,1);
    for k2 = 1:n
        dp = p(:,k2)-xyr(1:2,1);
        res(k2) = sqrt(dp’*dp)-xyr(3);
    end
end
```

An example of planar points are indicated by crosses illustrated in Fig. 6.9. The initial circle obtained by fitting three planar points among them and the final effect of circle fitting via (6.47) are demonstrated by the dashed circle and the solid circle respectively. The refined circle fits apparently better with the given planar points than the initial circle.

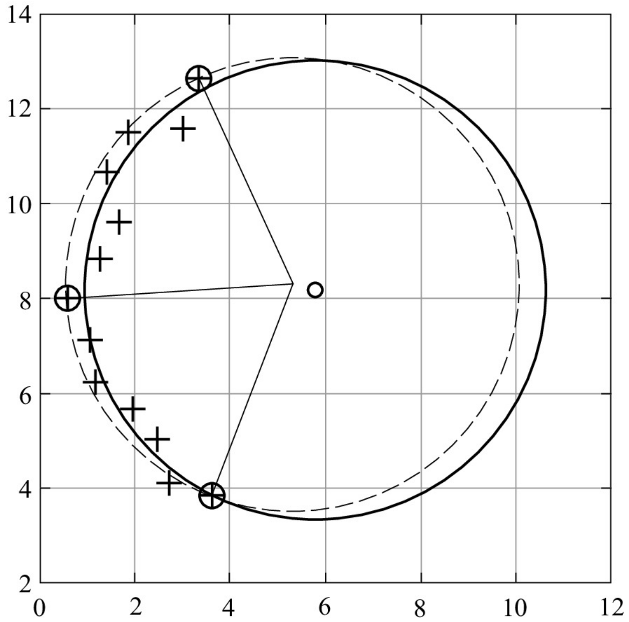


Fig. 6.9 Circle fitting with partial contour points

6.2.2 Recursive Linear LMS

Revisit the linear LMS problem formalized in (6.40) and (6.41). Suppose the linear parametric target model is

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I}}) = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_{\mathrm{I}} + b = [\boldsymbol{a} \quad b]^{\mathrm{T}}\boldsymbol{x}_{\mathrm{I}}^{A},$$

where the linear coefficients $\boldsymbol{a}$ and b form the parameter set $\boldsymbol{x}$ and $\boldsymbol{x}_{\mathrm{I}}^{A}$ denotes the augmented vector of $\boldsymbol{x}_{\mathrm{I}}$. Solving (6.41), i.e.

$$\boldsymbol{x} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| y_{\mathrm{O},i} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}_{\mathrm{I},i}^{A} \right\|_2^2$$

is equivalent to solving the linear equation

$$\begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},2}^{A\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},N}^{A\mathrm{T}} \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},2}^{\mathrm{T}} & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},N}^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} y_{\mathrm{O},1} \\ y_{\mathrm{O},2} \\ \vdots \\ y_{\mathrm{O},N-1} \\ y_{\mathrm{O},N} \end{bmatrix}. \tag{6.48}$$

Suppose one has already obtained the solution of (6.48), denoted as $\boldsymbol{x}_{[N]}$. Then imagine there is one more data pair $\{\boldsymbol{x}_{\mathrm{I},N+1}, y_{\mathrm{O},N+1}\}$ available now. The linear equation to-be-solved becomes a new one

$$\begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},2}^{A\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},N}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}} \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},2}^{\mathrm{T}} & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},N}^{\mathrm{T}} & 1 \\ \boldsymbol{x}_{\mathrm{I},N+1}^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} y_{\mathrm{O},1} \\ y_{\mathrm{O},2} \\ \vdots \\ y_{\mathrm{O},N-1} \\ y_{\mathrm{O},N} \\ y_{\mathrm{O},N+1} \end{bmatrix}. \tag{6.49}$$

Since the new linear equation (6.49) and the old linear equation (6.48) have absolutely the most part in common and since the solution $\boldsymbol{x}_{[N]}$ of the old linear equation (6.48) is already available, a question arises naturally: instead of solving the new linear equation (6.49) directly from scratch, can one take advantage of the available old solution $\boldsymbol{x}_{[N]}$ to facilitate computation of the new solution $\boldsymbol{x}_{[N+1]}$?

The answer is *yes*. For a generic N, denote

$$\boldsymbol{A}_N \equiv \begin{bmatrix} \boldsymbol{x}_{\mathrm{I},1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},2}^{A\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathrm{I},N-1}^{A\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{I},N}^{A\mathrm{T}} \end{bmatrix}, \quad \boldsymbol{B}_N \equiv \begin{bmatrix} y_{\mathrm{O},1} \\ y_{\mathrm{O},2} \\ \vdots \\ y_{\mathrm{O},N-1} \\ y_{\mathrm{O},N} \end{bmatrix}$$

which can also be denoted recursively as

$$\boldsymbol{A}_{N+1} = \begin{bmatrix} \boldsymbol{A}_N \\ \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}} \end{bmatrix}, \quad \boldsymbol{B}_{N+1} = \begin{bmatrix} \boldsymbol{B}_N \\ y_{\mathrm{O},N+1} \end{bmatrix}.$$

According to (F.7) presented in Sect. F.1 in Appendix F, the solution $\boldsymbol{x}_{[N]}$ is

$$\boldsymbol{x}_{[N]} = \left(\boldsymbol{A}_N^{\mathrm{T}} \boldsymbol{A}_N\right)^{-1} \boldsymbol{A}_N^{\mathrm{T}} \boldsymbol{B}_N \tag{6.50}$$

and the solution $\boldsymbol{x}_{[N+1]}$ is

$$\boldsymbol{x}_{[N+1]} = \left(\boldsymbol{A}_{N+1}^{\mathrm{T}} \boldsymbol{A}_{N+1}\right)^{-1} \boldsymbol{A}_{N+1}^{\mathrm{T}} \boldsymbol{B}_{N+1}. \tag{6.51}$$

Further denote $\boldsymbol{\Sigma}_N \equiv \left(\boldsymbol{A}_N^{\mathrm{T}} \boldsymbol{A}_N\right)^{-1}$. Then

$$
\begin{aligned}
\boldsymbol{A}_{N+1}^{\mathrm{T}}\boldsymbol{A}_{N+1} &= \begin{bmatrix}\boldsymbol{A}_{N}^{\mathrm{T}} & \boldsymbol{x}_{\mathrm{I},N+1}^{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_{N}\\ \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\end{bmatrix} = \boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{A}_{N} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}},\\
\boldsymbol{A}_{N+1}^{\mathrm{T}}\boldsymbol{B}_{N+1} &= \begin{bmatrix}\boldsymbol{A}_{N}^{\mathrm{T}} & \boldsymbol{x}_{\mathrm{I},N+1}^{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{B}_{N}\\ y_{\mathrm{O},N+1}\end{bmatrix} = \boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{B}_{N} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}y_{\mathrm{O},N+1}
\end{aligned}
$$

and

$$
\begin{aligned}
\boldsymbol{\Sigma}_{N+1} &= \left(\boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{A}_{N} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\right)^{-1} = \left(\boldsymbol{\Sigma}_{N}^{-1} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\right)^{-1}\\
&= \left[\left(\boldsymbol{I} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\right)\boldsymbol{\Sigma}_{N}^{-1}\right]^{-1} = \boldsymbol{\Sigma}_{N}\left(\boldsymbol{I} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\right)^{-1}\\
&= \boldsymbol{\Sigma}_{N}\left[\boldsymbol{I} + \sum_{k=1}^{\infty}(-1)^{k}\left(\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\right)^{k}\right]\\
&= \boldsymbol{\Sigma}_{N}\left[\boldsymbol{I} - \boldsymbol{x}_{\mathrm{I},N+1}^{A}\left(\sum_{k=1}^{\infty}(-1)^{k-1}\left(\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\right)^{k-1}\right)\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\right] \qquad (6.52)\\
&\qquad \text{(note that } \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A} \text{ is a scalar value)}\\
&= \boldsymbol{\Sigma}_{N}\left(\boldsymbol{I} - \frac{\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}\right) = \boldsymbol{\Sigma}_{N} - \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}},
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{x}_{[N+1]} &= \left(\boldsymbol{\Sigma}_{N} - \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}\right)\left(\boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{B}_{N} + \boldsymbol{x}_{\mathrm{I},N+1}^{A}y_{\mathrm{O},N+1}\right)\\
&= \boldsymbol{\Sigma}_{N}\boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{B}_{N} - \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{A}_{N}^{\mathrm{T}}\boldsymbol{B}_{N}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}} +\\
&\qquad \left(\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A} - \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}\right)y_{\mathrm{O},N+1} \qquad (6.53)\\
&= \boldsymbol{x}_{[N]} - \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{x}_{[N]}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}} + \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}y_{\mathrm{O},N+1}\\
&= \boldsymbol{x}_{[N]} + \frac{\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A}}\left(y_{\mathrm{O},N+1} - \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{x}_{[N]}\right).
\end{aligned}
$$

To better highlight the recursive characteristic of (6.52) and (6.53), formalize them equivalently as

$$
\begin{aligned}
\boldsymbol{S}: &= \boldsymbol{\Sigma}_{N}\boldsymbol{x}_{\mathrm{I},N+1}^{A},\\
\boldsymbol{K}: &= \frac{\boldsymbol{S}}{1+\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{S}}, \qquad (6.54\text{a})\\
\boldsymbol{\Sigma}_{N+1} &= \boldsymbol{\Sigma}_{N} - \boldsymbol{K}\boldsymbol{S}^{\mathrm{T}},
\end{aligned}
$$

$$
\boldsymbol{x}_{[N+1]} = \boldsymbol{x}_{[N]} + \boldsymbol{K}\left(y_{\mathrm{O},N+1} - \boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}\boldsymbol{x}_{[N]}\right). \qquad (6.54\text{b})
$$

The set of parameters $\{\boldsymbol{x}_{[N]}, \boldsymbol{\Sigma}_N\}$ is updated recursively as N increases. This rather resembles the practice of recursive estimation [23]. In fact, if one treats $\{\boldsymbol{x}_{[N]}, \boldsymbol{\Sigma}_N\}$ as the state estimate, i.e. the state vector with the state covariance and treats $\boldsymbol{x}_{\mathrm{I},N+1}^{A\mathrm{T}}$ as the measurement matrix at current index, then (6.54) is right a special update formalism of the well-known *Kalman filter*.

6.2.3 Regularized LMS

Regularization refers to the methodology of imposing a priori information or desired characteristics on solutions of numeric computation and optimization [24, 25]. It is usually realized in the form of optimizing certain specially designed **loss function**. A representative method of regularization is called *ridge regression* [26].

The method is formalized as

$$\boldsymbol{x} \ = \underset{\boldsymbol{x}}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_{\mathrm{O},i} - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},i})\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2 \right] \tag{6.55}$$

where λ is certain regularizing coefficient. If λ is set to 0, then (6.55) is reduced to (6.39). In other words, the conventional least mean squares method can be regarded as a special case of the regularized least mean squares method.

6.3 Unknown Feature-Model Relationship Oriented Model Fitting

In this sub-section the author focuses on unknown feature-model relationship oriented model fitting.

6.3.1 Hough Transform

The Hough transform [27] method is a classical method for model fitting when feature-model relationship is unknown. Suppose one needs to fit lines to a set of planar points, as illustrated in Fig. 6.10. In this example, a number of feature points form a line pattern, but not all feature points belong to the line. The line pattern and its relationship with feature points are intuitive to humans. However, without any a priori knowledge of which feature points are associated with the line pattern, how can the computer retrieve the line pattern?

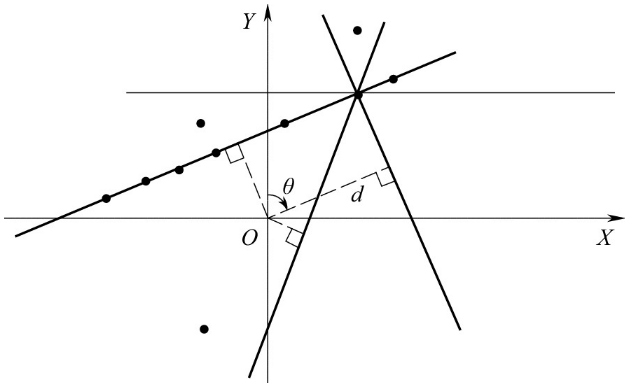


Fig. 6.10 Line Hough transform

One can apply the Hough transform method to handle the problem. The idea is to consider the *configuration space* Ω of lines located in the target plane area. Then let each feature point vote for lines in the configuration space: it gives a vote to each line associated with it, but gives no vote to any line not associated with it. It is true that the configuration space consists of uncountable number of lines. For feasibility in practice, one can discretize the configuration space into a finite set of lines. More specifically, parametrize a generic line by the normal distance d and the normal angle θ. Suppose the maximum of the distances of feature points to the origin is $d_{\max}$. The continuous configuration space Ω is

$$\Omega = \{(d, \theta) | -d_{\max} \leqslant d \leqslant d_{\max}, 0 \leqslant \theta < 2\pi\}$$

and a discretized configuration space Ω_{D} can be set to

$$\Omega_{\mathrm{D}} = \{(d, \theta) | d \in [-d_{\max} : \Delta d : d_{\max}], \theta \in [0 : \Delta\theta : 2\pi)\},$$

where Δd and $\Delta\theta$ denote the discrete normal distance interval and the discrete normal angle interval respectively.

Once the feature points have finished their voting for lines in the discretized configuration space, then retrieve lines with high votes and treat them as intended line patterns. For the example illustrated in Fig. 6.10, most lines in the discretized configuration space Ω_{D} have no vote at all, some lines in Ω_{D} have one vote, some lines in Ω_{D} have two votes etc. In contrast, there is a line that has seven votes (the highest vote) and this line is treated as the intended line pattern.

The methodology of the Hough transform is general ,[9] not limited to line patterns retrieval. Besides the line Hough transform, the circle Hough transform is also available. For example, the circle Hough transform can be used to retrieve circle patterns that correspond to round traffic signs in the image, as demonstrated in Fig. 6.11.

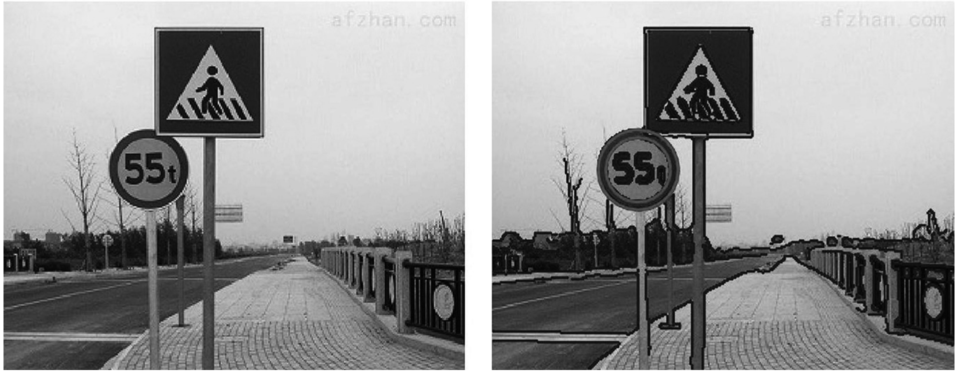


Fig. 6.11 Circle Hough transform for round traffic sign detection

6.3.2 Expectation Maximization (EM)

Given a group of mutually exclusive model patterns that form together a kind of model of general and flexible representation ability, known as a **mixture model**. View features as having been generated by the model patterns (i.e. from the *generative model* perspective). Suppose there are m model patterns parametrized as

$$\boldsymbol{\Omega} \equiv \{\Omega_1, \ldots, \Omega_m; \pi_1, \ldots, \pi_m\}. \tag{6.56}$$

More specifically, for each $j \in \{1, \ldots, m\}$, Ω_j denotes the set of parameters that characterize the *j*-th model pattern, and π_j denotes the probability of choosing the *j*-th model pattern to generate a feature. $\{\pi_1, \ldots, \pi_m\}$ are called **mixture weights** and they satisfy the normalization condition

$$\pi_1 + \cdots + \pi_m = \sum_{j=1}^{m} \pi_j = 1.$$

Suppose there are *N* features F_1, F_2, ..., F_N. For a generic feature F_i ($i \in \{1, \ldots, N\}$), define the *indicator vector*

$$\boldsymbol{I}_i \equiv [I_{i1} \quad \cdots \quad I_{im}]$$

where the *indicator variable* I_{ij} denotes whether the feature F_i is generated by the *j*-th model pattern, namely

$$I_{ij} = \begin{cases} 1, & \text{if feature } F_i \text{ is generated by } \Omega_j \\ 0, & \text{otherwise} \end{cases}$$

The indicator variables are *hidden variables* that are unknown.

Consider the maximum likelihood estimation with hidden indicator variables

$$\{\boldsymbol{\Omega}, \boldsymbol{I}\} = \underset{\boldsymbol{\Omega}, \boldsymbol{I}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(F_i, \boldsymbol{I}_i | \boldsymbol{\Omega}) \tag{6.57}$$

where $\boldsymbol{I} \equiv \{\boldsymbol{I}_1, \ldots, \boldsymbol{I}_N\}$. Note that

$$p(F_i, \boldsymbol{I}_i | \boldsymbol{\Omega}) = p(F_i | \boldsymbol{I}_i, \boldsymbol{\Omega}) p(\boldsymbol{I}_i | \boldsymbol{\Omega}) = p(F_i | \Omega_{j|I_{ij}=1}) \pi_{j|I_{ij}=1}.$$

Since

$$[p(F_i|\Omega_j)\pi_j]^{I_{ij}} = \begin{cases} p(F_i|\Omega_{j|I_{ij}=1})\pi_{j|I_{ij}=1} & \text{if } \mathrm{I_{ij}} = 1 \\ 1 & \text{if } \mathrm{I_{ij}} = 0 \end{cases}$$

one has

$$p(F_i, \boldsymbol{I}_i | \boldsymbol{\Omega}) = \textstyle\prod_{j=1}^{m} [p(F_i|\Omega_j)\pi_j]^{I_{ij}}. \tag{6.58}$$

Substitute (6.58) into (6.57) and obtain

$$\{\boldsymbol{\Omega}, \boldsymbol{I}\} = \underset{\boldsymbol{\Omega}, \boldsymbol{I}}{\operatorname{argmax}} \sum_{i=1}^{N} \sum_{j=1}^{m} I_{ij} [\log p(F_i|\Omega_j) + \log \pi_j] \tag{6.59}$$

As $\boldsymbol{I} \equiv \{\boldsymbol{I}_1, \ldots, \boldsymbol{I}_N\} \equiv \{\{I_{11}, \ldots, I_{1m}\}, \ldots, \{I_{N1}, \ldots, I_{Nm}\}\}$ are hidden variables that are unknown, to make (6.59) tractable in practice, replace each I_{ij} by its expectation

$$I_{ij}^E = E[I_{ij}].$$

Suppose the mixture model parameter set $\boldsymbol{\Omega}$ is fixed, then for a generic feature F_i, the expectation $\boldsymbol{I}_i^E \equiv \{I_{i1}^E, \dots, I_{im}^E\}$ may be estimated as the a posteriori distribution conditioned on $\boldsymbol{\Omega}$ and F_i

$$I_{ij}^E = p(I_{ij}|F_i, \boldsymbol{\Omega}) = \frac{p(F_i|I_{ij},\boldsymbol{\Omega})p(I_{ij}|\boldsymbol{\Omega})}{p(F_i)} = \frac{p(F_i|\Omega_j)\pi_j}{\sum_{j'=1}^{m} p(F_i|\Omega_{j'})\pi_{j'}} \tag{6.60}$$

where $j \in \{1, \dots, m\}$. Further suppose the expectation $\boldsymbol{I}^E$ is fixed, then (6.59) is reduced to an optimization problem without hidden indicator variables

$$\boldsymbol{\Omega} = \underset{\boldsymbol{\Omega}}{\mathrm{argmax}} \sum_{i=1}^{N} \sum_{j=1}^{m} I_{ij}^E [\log p(F_i|\Omega_j) + \log \pi_j]. \tag{6.61}$$

Applied in alternating way, (6.60) and (6.61) naturally formalize an iterative solving of (6.59), which is called the **expectation maximization** method [29]. Each iteration consists of two steps, namely the **E-step** that is associated with (6.60) and the **M-step** that is associated with (6.61).

Line fitting with outliers

To facilitate understanding of the working mechanism of the expectation maximization method, one may consider the problem of line fitting with outliers: given a set of feature points on the plane, some of which form a line pattern whereas the others of which are scattered randomly within the target area. Which feature points are associated with the line pattern and which ones are randomly scattered are unknown; in other words, the relationship between the line model pattern and feature points is unknown.

To handle the problem, consider a mixture model that consists of two model patterns: one is the line model pattern parametrized as

$$\Omega_1: \quad y = ax + b + \gamma \quad \text{with } \gamma \sim N(0, \sigma)$$

and the other is the uniformly random distribution model pattern

$$\Omega_2: \quad x \sim U[0, W], \quad y \sim U[0, H]$$

where W and H denote the width and height of the target area respectively.

E-step: Given a generic feature point (x, y), apply (6.60) as

$$I_{(x,y)1}^E = \frac{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-ax-b)^2}{2\sigma^2}} \pi_1}{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-ax-b)^2}{2\sigma^2}} \pi_1 + \frac{1}{WH} \pi_2} = \frac{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-ax-b)^2}{2\sigma^2}} \pi_1}{\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-ax-b)^2}{2\sigma^2}} \pi_1 + \frac{1}{WH} (1-\pi_1)}, \tag{6.62a}$$

$$I^E_{(x,y)2} = \frac{\frac{1}{WH}\pi_2}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(y-ax-b)^2}{2\sigma^2}}\pi_1+\frac{1}{WH}\pi_2} = \frac{\frac{1}{WH}(1-\pi_1)}{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(y-ax-b)^2}{2\sigma^2}}\pi_1+\frac{1}{WH}(1-\pi_1)}. \qquad (6.62b)$$

M-step: Given fixed $I^E_{(x_i,y_i)1}$ and $I^E_{(x_i,y_i)2}$ where $i \in \{1,\dots,N\}$, apply (6.61) as

$$\begin{aligned}\{a,b,\pi_1\} = \quad & \underset{a,b,\pi_1}{\operatorname{argmax}} \sum_{i=1}^{N}\Bigg[I^E_{(x_i,y_i)1}\left(\log\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(y_i-ax_i-b)^2}{2\sigma^2}}+\log\pi_1\right) \\ & +I^E_{(x_i,y_i)2}\left(\log\tfrac{1}{WH}+\log\left(1-\pi_1\right)\right)\Bigg] \\ = \quad & \underset{a,b,\pi_1}{\operatorname{argmax}} \sum_{i=1}^{N}\Bigg[-I^E_{(x_i,y_i)1}\frac{(y_i-ax_i-b)^2}{2\sigma^2}+ \\ & I^E_{(x_i,y_i)1}\log\pi_1+I^E_{(x_i,y_i)2}\log\left(1-\pi_1\right)\Bigg]\end{aligned}$$

which can be decomposed into two sub-optimization problems

$$\{a,b\} \quad = \underset{a,b}{\operatorname{argmin}}\sum_{i=1}^{N}I^E_{(x_i,y_i)1}(y_i-ax_i-b)^2 \qquad (6.63a)$$

$$\pi_1 \quad = \underset{\pi_1}{\operatorname{argmax}}\left(\sum_{i=1}^{N}I^E_{(x_i,y_i)1}\right)\log\pi_1+\left(\sum_{i=1}^{N}I^E_{(x_i,y_i)2}\right)\log\left(1-\pi_1\right). \qquad (6.63b)$$

The solution of the first equation of (6.63) is the solution of the linear equation

$$\begin{bmatrix}\sqrt{I^E_{(x_1,y_1)1}}x_1 & \sqrt{I^E_{(x_1,y_1)1}} \\ [9pt]\sqrt{I^E_{(x_2,y_2)1}}x_2 & \sqrt{I^E_{(x_2,y_2)1}} \\ [5pt]\vdots & \vdots \\ [5pt]\sqrt{I^E_{(x_N,y_N)1}}x_N & \sqrt{I^E_{(x_N,y_N)1}}\end{bmatrix}\begin{bmatrix}a \\ b\end{bmatrix} = \begin{bmatrix}\sqrt{I^E_{(x_1,y_1)1}}y_1 \\ [9pt]\sqrt{I^E_{(x_2,y_2)1}}y_2 \\ [5pt]\vdots \\ [5pt]\sqrt{I^E_{(x_N,y_N)1}}y_N\end{bmatrix}$$

and the solution of the second equation of (6.63) is

$$\pi_1 = \frac{\sum_{i=1}^{N} I^E_{(x_i,y_i)1}}{\sum_{i=1}^{N} I^E_{(x_i,y_i)1} + \sum_{i=1}^{N} I^E_{(x_i,y_i)2}} = \frac{\sum_{i=1}^{N} I^E_{(x_i,y_i)1}}{N}.$$

Matlab code for demonstrating expectation maximization based line fitting with outliers is given as follows.

LineFittingWithOutliers.m

```
%% Demonstrate line fitting with outliers
% Generate synthetic data
W = 100; H = 100; a = 0.5; b = H/2-a*(W/2); s = 0.5;
iterN = 60; aorg = a; borg = b; sorg = s;
xy = rand(2,100); xr = xy(1,:)*W; yr = xy(2,:)*H;
x = xr; y = a*x+b+random(’norm’,0,s,1,length(x));
x = [x, xr]; y = [y, yr]; N = length(x);

% Expectation maximization
I1 = ones(1,N)/N; a = 2*rand()-1; b = rand()*H; p1 = rand();
for k = 1:iterN
    if (k<=iterN/2) s = 10*sorg; else s = sorg; end
    % E-step
    tmp = (1/(sqrt(2*pi)*s))*exp(-(y-a*x-b).^2/(2*s^2))*p1;
    I1 = tmp./(tmp+(1/(W*H))*(1-p1));
    % M-step
    AB = diag(sqrt(I1))*[x’, ones(size(x’)), y’];
    tmp = AB(:,1:2)\AB(:,3); a = tmp(1); b = tmp(2); p1 = sum(I1
    )/N;
    fprintf(’Round %d : [a, b, p1] = [%f, %f, %f]\n’, k, a, b,
    p1);
end
figure(1), hold on; plot(x,y,’ob’,’LineWidth’,2);
line([0,W],[b,a*W+b],’Color’,’r’,’LineWidth’,3);
hold off; axis equal; xlim([0,W]); ylim([0,H]); axis off;
```

The effect of line fitting with outliers is demonstrated in Fig. 6.12.

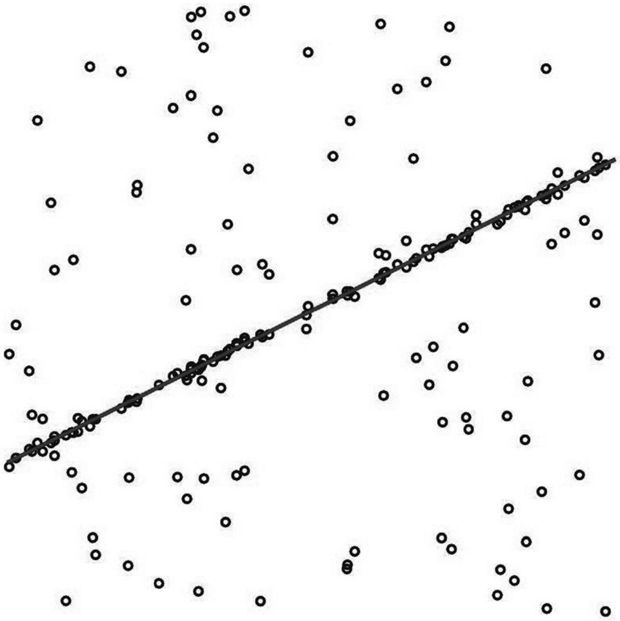


Fig. 6.12 Expectation maximization based line fitting with outliers

Discussion

The expectation maximization method is not guaranteed to converge to the correct solution and is inclined to get stuck in certain local minimum if an initial estimate proper enough is not provided. To alleviate the problem, a tentative technique is to try the expectation maximization method with various initial estimates and keep the best fitting—This is rather like the RANSAC method that will be presented later—For concrete practical applications, one may also resort to ad hoc techniques to handle the problem of wrong convergence. For the demonstrated example of line fitting with outliers, one can adopt an ad hoc technique that consists in varying the standard deviation of the line model pattern adaptively. This idea may somehow be generalized, namely to vary certain configurations of model patterns adaptively.

6.3.3 Random Sample Consensus (RANSAC)

The **RANdom SAmple Consensus** (RANSAC) method [30] is a general and elegant method of unknown feature-model relationship oriented model fitting. The basic idea is to

- Randomly select sample features to fit a target model.
- Evaluate the number of all features that are consistent with the model fitted by sample features.
- Repeat above sampling and evaluation process.
- Finally take the best model fitting result namely the one that is consistent with the largest number of features.

The RANSAC method is rather a methodology that can potentially be applied to handle a wide range of problems in practice. Concrete applications of the RANSAC method will be demonstrated in Chap. 7.

Required number of sampling rounds

Suppose each sampling round takes m features and a qualified fitting of the target model will be formed if and only if the m sample features are all qualified. Further suppose the probability of a sample feature being qualified is w $(0 < w < 1)$. Let n

denote the number of sampling rounds to obtain a qualified fitting of the target model.

The expectation of n can be computed as

$$\begin{aligned} E[n] = \quad & 1 \cdot P\{\text{qualified fitting by one sampling round}\} + \\ & 2 \cdot P\{\text{qualified fitting by two sampling rounds}\} + \\ & 3 \cdot P\{\text{qualified fitting by three sampling rounds}\} + \cdots \end{aligned}$$

namely

$$\begin{aligned} E[n] & = \sum_{k=1}^{\infty} k \cdot P\{\text{qualified fitting by k sampling rounds}\} = \sum_{\mathrm{k}=1}^{\infty} \mathrm{k} \cdot \left(1 - \mathrm{w}^{\mathrm{m}}\right)^{\mathrm{k}-1} \mathrm{w}^{\mathrm{m}} \\ & = w^m \left[\sum_{k=1}^{\infty} k \left(1 - w^m\right)^{k-1} \right] = w^m \cdot \frac{1}{\left[1 - \left(1 - w^m\right)\right]^2} = \frac{1}{w^m}. \end{aligned} \tag{6.64}$$

Derivation of (6.64) resorts to the following equality of infinite power series

$$\sum_{k=1}^{\infty} k \cdot x^{k-1} = \frac{1}{(1-x)^2}$$

which can be proved as follows

$$\begin{aligned} \sum_{k=1}^{\infty} k \cdot x^{k-1} & = \sum_{k=1}^{\infty} \frac{\mathrm{d}}{\mathrm{d}x}\left(x^k\right) = \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{k=1}^{\infty} x^k\right) = \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{k=0}^{\infty} x^k - 1\right) \\ & = \tfrac{\mathrm{d}}{\mathrm{d}x}\left(\tfrac{1}{1-x} - 1\right) = \tfrac{1}{(1-x)^2}. \end{aligned}$$

The variance of n can be computed as

$$\begin{aligned} \mathrm{Var}(\mathrm{n}) & = E\left[n^2\right] - E[n]^2 = \sum_{k=1}^{\infty} k^2 \cdot \left(1 - w^m\right)^{k-1} w^m - \left(\frac{1}{w^m}\right)^2 \\ & = \tfrac{1 + (1 - w^m)}{\left[1 - (1 - w^m)\right]^3} \cdot w^m - \tfrac{1}{w^{2m}} = \tfrac{1 - w^m}{w^{2m}} \end{aligned} \tag{6.65}$$

and hence the standard deviation of n is computed as

$$\sqrt{\mathrm{Var}(\mathrm{n})} = \tfrac{\sqrt{1 - w^m}}{w^m}. \tag{6.66}$$

Derivation of (6.65) resorts to the following equality of infinite power series

$$\sum_{k=1}^{\infty} k^2 \cdot x^{k-1} = \frac{1+x}{(1-x)^3}$$

which can be proved as follows

$$
\begin{aligned}
\sum_{k=1}^{\infty} k^2 \cdot x^{k-1} &= \sum_{k=1}^{\infty}(k+1)k \cdot x^{k-1} - \sum_{k=1}^{\infty} k \cdot x^{k-1} \\
&= \frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(\sum_{k=1}^{\infty} x^{k+1}\right) - \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{k=1}^{\infty} x^{k}\right) \\
&= \tfrac{\mathrm{d}^2}{\mathrm{d}x^2}\left(\tfrac{1}{1-x} - x - 1\right) - \tfrac{\mathrm{d}}{\mathrm{d}x}\left(\tfrac{1}{1-x} - 1\right) = \tfrac{1+x}{(1-x)^3}.
\end{aligned}
$$

Equations (6.64) and (6.66) provide a guide to estimate a number of sampling rounds required to probabilistically guarantee appearance of at least one qualified fitting of the target model.

References

1. M. Wertheimer, Untersuchungen zur lehre von der gestalt ii. Psychol. Forsch. **4**, 301–350 (1923) [Crossref]
2. H. Steinhaus, Sur la division des corps matériels en parties. Bulletin de l’Académie Polonaise des Sciences **4**(12), 801–804 (1956)
3. S. Lloyd, Least squares quantization in pcm. IEEE Trans. Inform. Theory **28**(2), 129–137 (1982) [ADS][MathSciNet][Crossref][zbMATH]
4. L. Kaufman, P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis* (Wiley, 1990)
5. K. Fukunaga, L. Hostetler, The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Inform. Theory **21**(1), 32–40 (1975) [ADS][MathSciNet][Crossref][zbMATH]
6. Y. Cheng, Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Mach. Intell. **17**(8), 790–799 (1995) [ADS][Crossref]
7. D. Comaniciu, P. Meer, Mean shift: a robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Mach. Intell. **24**(5), 603–619 (2002) [ADS][Crossref]
8. M. Rosenblatt, Remarks on some nonparametric estimates of a density function. Ann. Math. Statist. **27**(3), 832–837 (1956) [MathSciNet][Crossref][zbMATH]
9. E. Parzen, On estimation of a probability density function and mode. Ann. Math. Statist. **33**(3), 1065–1076 (1962) [ADS][MathSciNet][Crossref][zbMATH]
10. W. Tao, H. Jin, Y. Zhang. Color image segmentation based on mean shift and normalized cuts. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **37**(5), 1382–1389 (2007)
11. J. Jang, H. Jiang. Meanshift++: extremely fast mode-seeking with applications to segmentation and object tracking, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021) pp. 4102–4113
12. J. Schanda (Ed.). *Colorimetry: Understanding the CIE System* (Wiley, 2007)
13. J. Shi, J. Malik, Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000) [ADS][Crossref]
14. D. West. *Introduction to Graph Theory* (Pearson Education, 2018)
15. G. Stewart. *Matrix Algorithms: (Volume II) Eigensystems* (SIAM, 2001)

16. G. Iannizzotto, L. Vita, Fast and accurate edge-based segmentation with no contour smoothing in 2-d real images. IEEE Trans. Image Process. **9**(7), 1232–1237 (2000)
[ADS][Crossref][PubMed]
17. M. Wani, B. Batchelor, Edge-region-based segmentation of range images. IEEE Trans. Pattern Anal. Mach. Intell. **16**(3), 314–319 (1994)
[ADS][Crossref]
18. Y. Boykov, G. Funka-Lea, Graph cuts and efficient n-d image segmentation. Int. J. Comput. Vis. **70**(2), 109–131 (2006)
[Crossref]
19. S. Chew, N. Cahill. Semi-supervised normalized cuts for image segmentation, in *IEEE International Conference on Computer Vision* (2015), pp. 1716–1723
20. M. Tang, A. Djelouah, F. Perazzi, Y. Boykov, C. Schroers, Normalized cut loss for weakly-supervised cnn segmentation, in *IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 1818–1827
21. X. Wang, R. Girdhar, S. Yu, I. Misra, Cut and learn for unsupervised object detection and instance segmentation, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 3124–3134
22. S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, D. Terzopoulos, Image segmentation using deep learning: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **44**(7), 3523–3542 (2022)
[ADS][PubMed]
23. H. Li., *Fundamentals and Applications of Recursive Estimation Theory* (Shanghai Jiao Tong University Press, 2022)
24. V. Arsenin, A. Tikhonov (Tychonoff). *Solutions of ill-Posed Problems* (Winston & Sons, 1977)
25. D. Phillips, A technique for the numerical solution of certain integral equations of the first kind. J. ACM **9**(1), 84–97 (1962)
[MathSciNet][Crossref][zbMATH]
26. A. Hoerl, R. Kennard, Ridge regression: applications to nonorthogonal problems. Technometrics **12**(1), 69–82 (1970)
[Crossref][zbMATH]
27. R. Duda, P. Hart, Use of the hough transformation to detect lines and curves in pictures. Commun. ACM **15**(1), 11–15 (1972)
[Crossref][zbMATH]
28. R. Bellman, The theory of dynamic programming. Bull. Am. Math. Soc. **60**(6), 503–515 (1954)
[MathSciNet][Crossref][zbMATH]
29. A. Dempster, N. Laird, D. Rubin. Maximum likelihood from incomplete data via the em algorithm. J. R. Statist. Soc. Ser B (Methodol.) **39**(1):1–38 (1977)
30. M. Fischler, R. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
[MathSciNet][Crossref]

Footnotes

[1] Originally in [2], an optimization problem equivalent to the optimization problem (6.1) was formalized with a concrete *mechanics* interpretation, namely to search for the optimal “division d’un corps” or “décomposition” that minimizes the summed “moment d’inertie”.

[2] The mechanism of adaptive clustering decision demonstrated in the given example is intentionally simplified to facilitate readers’ understanding. For more knowledge on such aspect especially *hierarchical clustering* via either *agglomerative analysis* or *divisive analysis*, readers may refer to [4]. By the way, the qualitative expression “two smaller clusters which are apart considerably ... and are rather self-concentrated respectively ...” can be more formally paraphrased with the help of quantitative terms *inter-cluster variance* and *intra-cluster variance*.

[3] The feature space had better be scaled consistently for the various dimensions.

[4] Typical works that provide various aspects of such considerations include [7, 10, 11]. For example, one may resort to a spatial-range joint domain kernel [7], to a cooperation framework that augments the mean shift method with another clustering method [10], to a special design of grid based implementation [11], etc.

[5] One may follow the mathematically more standard term *minimum cut problem* in the context of *combinatorial optimization*, and treat the problem of minimizing (6.20) still as a kind of minimum cut problem, but just with a variant definition of the *cut cost*.

[6] Even without theoretical analysis, it rather makes sense to know intuitively that there must be a redundant variable, because an arbitrarily chosen vertex individually can be assigned to whatever sub-graph in the bipartition. It is after assignment of the vertex is determined that assignment of other vertices can also be determined accordingly: some other vertices are assigned to the same sub-graph as the vertex, whereas the remaining vertices are assigned to the other sub-graph. The key point consists in determining which other vertices and the vertex are assigned to the same sub-graph, but to which same sub-graph they are assigned, be it $\boldsymbol{G}_1$ or $\boldsymbol{G}_2$, will bring no essential difference to the bipartition itself. So the indicator variable of the arbitrarily chosen vertex is redundant.

[7] If one did expect more data to play a more influential role for special applications, then the logic would be different.

[8] If circle fitting can be done for planar points that represent the partial contour, then circle fitting can apparently be done as well for planar points that represent the entire contour of a circle.

[9] In as early as 1950s, *Bellman* pointed out a problem that is nowadays commonly coined as the *curse of dimensionality*, namely a problem of excessive dimensionality causing unaffordable processing price [28]. The curse of dimensionality occurs in many domains such as numerical computations, statistics, combinatorics, data mining, and machine learning. The Hough transform method tends to suffer from the curse of dimensionality as well if the configuration space is of a high dimensionality.

© Shanghai Jiao Tong University Press 2025
H. Li, *Geometry Fundamentals of Computer Vision*
https://doi.org/10.1007/978-981-96-7344-5_7

7. Applications

Hao Li[1] ✉
(1) Shanghai Jiao Tong University, Shanghai, China

7.1 Ground Object Detection
7.2 Virtual Reality and Augmented Reality
7.3 Cooperative Augmented Reality: See-Through Effect
7.4 Image Stitching
7.5 Special Tag Based Visual Localization
7.6 Visual SLAM
References

Geometry fundamentals of computer vision play an important role in many interesting and useful applications, some of which are presented in this Chapter.

7.1 Ground Object Detection

Ground is probably the most important environment part in daily life. Many practical applications involve detection of ground objects. For example, lane detection [1], as demonstrated in Fig. 7.1, is indispensable to intelligent vehicles which have been arousing more and more research and commercial interests. Lane detection is also indispensable to lane keeping oriented advanced driver assistance system (ADAS).

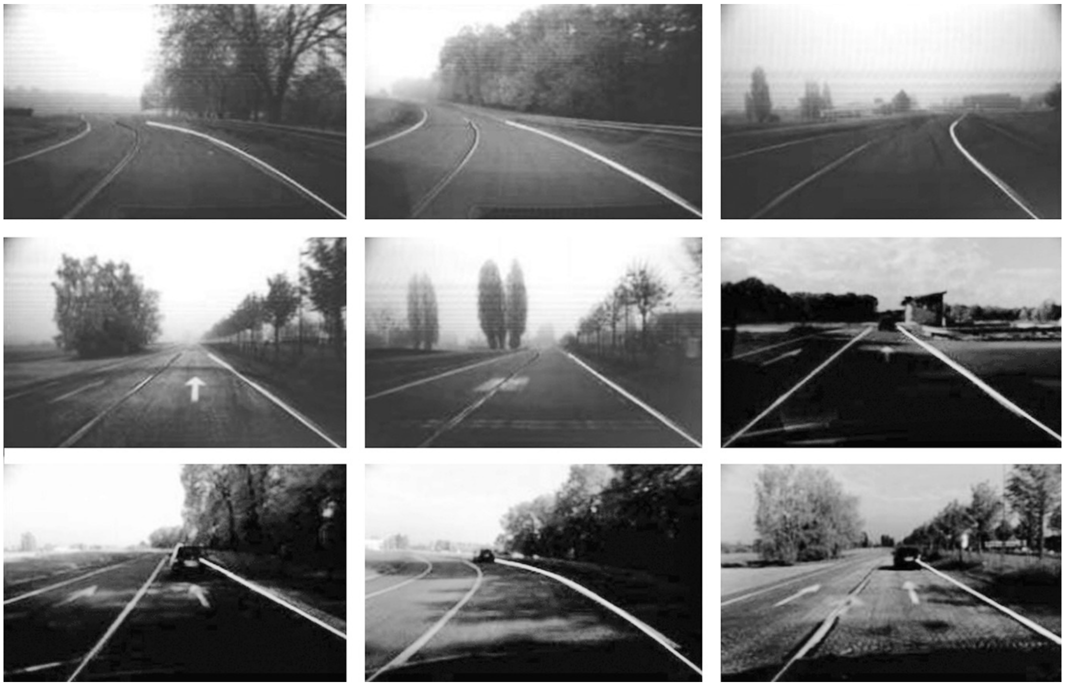


Fig. 7.1 Lane detection [1]

For an intelligent vehicle, to facilitate image processing of ground objects such as lane marks, one can establish a special world coordinates system namely the *vehicle-ground coordinates system* as follows: Attach the vehicle-ground coordinates system $O_\mathrm{w}-X_\mathrm{w}-Y_\mathrm{w}-Z_\mathrm{w}$ to the intelligent vehicle, with the $O_\mathrm{w}-X_\mathrm{w}-Y_\mathrm{w}$ plane located on the ground surface, the X_w axis pointing forward along the longitudinal direction, the Y_w axis pointing left along the lateral direction, and the Z_w axis pointing upward. The vehicle-ground coordinates system is illustrated in Fig. 7.2. Since the vehicle-ground coordinates system is attached to and moving with the intelligent vehicle, it describes world coordinates of objects with respect to the intelligent vehicle instead of a global reference.

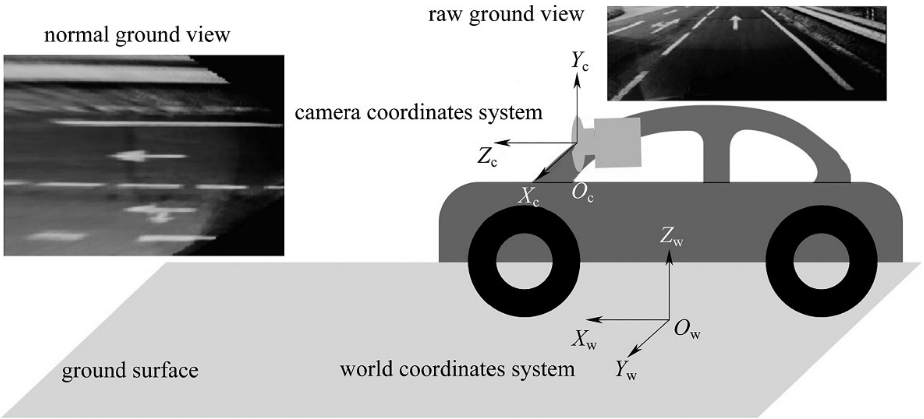

raw ground view
normal ground view
Y_c
camera coordinates system
Z_c
X_c
O_c
Z_w
X_w
O_w
Y_w
ground surface
world coordinates system

Fig. 7.2 Vehicle-ground coordinates system

Suppose the perspective mapping between the vehicle-ground coordinates system and the image coordinates system is

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}] \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \\ 1 \end{bmatrix}. \quad (7.1)$$

Here, the camera intrinsic parameters can still be calibrated using the *Zhang*'s chessboard pattern method presented in Chap. 5. Once the camera intrinsic parameters are calibrated, one can theoretically calibrate the camera extrinsic parameters with a chessboard posed on the ground surface. However, this does not work in practice because a chessboard is too small to cover enough image part.

Despite infeasibility of using a small chessboard to calibrate the camera extrinsic parameters with respect to the vehicle-ground coordinates system, the theoretical principles formalized in (5.2) and (5.40) can still be adopted. For the calibration practice, one just needs to replace the small chessboard pattern by a large ground surface pattern, such as illustrated in Fig. 7.3.

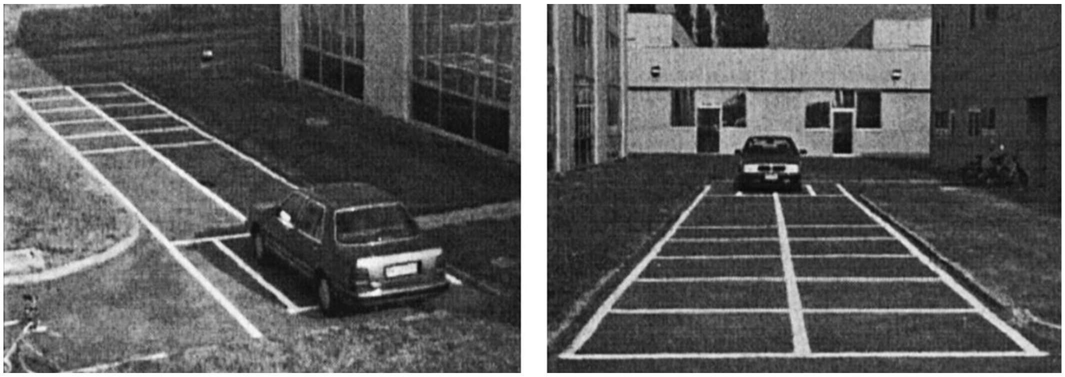


Fig. 7.3 Large ground surface pattern for calibrating the camera extrinsic parameters [2]

With the help of the large ground surface pattern, first calibrate the homography $\boldsymbol{H}$ between the ground surface plane, i.e. the $O_{\mathrm{w}} - X_{\mathrm{w}} - Y_{\mathrm{w}}$ plane and the image plane. Then resort to (5.2) and (5.40) to compute the rotation matrix $\boldsymbol{R}_{\mathrm{wc}}$ (or the rotation vector $\boldsymbol{w}_{\mathrm{wc}}$) and the translation vector $\boldsymbol{T}_{\mathrm{wc}}$:

$$\begin{aligned}
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1 :&= \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}\|_2},\\
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2 :&= \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}\|_2},\\
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_3 :&= (\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1) \times (\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2),\\
\boldsymbol{w}_{\mathrm{wc}} &= \boldsymbol{R}^{-1}(\boldsymbol{R}_{\mathrm{wc}}) = \boldsymbol{R}^{-1}([\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_3]),\\
\boldsymbol{R}_{\mathrm{wc}} :&= \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}) \qquad (\text{update the rotation matrix}),\\
\lambda &= \sqrt{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}\|_2 \cdot \|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}\|_2},\\
\boldsymbol{T}_{wc} &= \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,3},
\end{aligned}$$

where $\boldsymbol{R}(\cdot)$ denotes the rotation function that maps a rotation vector to its associated rotation matrix and $\boldsymbol{R}^{-1}(\cdot)$ denotes the inverse rotation function that maps a (quasi)rotation matrix to its associated rotation vector via the method presented in Sect. 5.2.1 in Chap. 5.

One can further refine the estimate of camera extrinsic parameters via (5.38) and (5.39), namely to solve a group of nonlinear equations in terms of free variables $\{\boldsymbol{w}_{\mathrm{wc}}, \boldsymbol{T}_{\mathrm{wc}}\}$:

$$\frac{\begin{bmatrix}\boldsymbol{e}_1^{\mathrm{T}}\\ \boldsymbol{e}_2^{\mathrm{T}}\end{bmatrix}\boldsymbol{A}[\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}) \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w},l}^{A}}{\boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{A}[\boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}) \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w},l}^{A}} = \boldsymbol{P}_{\mathrm{i},l},$$

where $\{\boldsymbol{P}_{\mathrm{w},l}\}$ and $\{\boldsymbol{P}_{\mathrm{i},l}\}$ ($l \in \{1, \ldots, m\}$) denote control points provided by the large ground surface pattern.

Once the camera intrinsic and extrinsic parameters are calibrated offline in advance, the bijective relationship between the ground surface and the image plane is determined and can be used during online operation of the intelligent vehicle. For lane detection, the raw ground view can be projected onto the ground surface plane, i.e. the $O_{\mathrm{w}} - X_{\mathrm{w}} - Y_{\mathrm{w}}$ plane via *inverse perspective mapping* to form the normal ground view, as illustrated in Fig. 7.2. The inverse perspective mapping method has already been presented in Sect. 4.3.3 in Chap. 4.

Processing of ground objects on the normal ground view instead of the raw ground view has the merit of enabling the processing results to be related to actual physical properties of ground objects in the real world and hence enabling general *a priori* knowledge of ground objects to be exploited for detection of them. For example, one can take advantage of general *a priori* knowledge such as normal ranges of lane mark length, lane width, and lane mark arrangement regularity to sift out outliers of lane mark features.

In practice, the $O_{\mathrm{w}} - X_{\mathrm{w}} - Y_{\mathrm{w}}$ plane of the vehicle-ground coordinates system is usually slightly inconsistent with the ground surface plane, due to the pitch angle of the intelligent vehicle. An adaptive mechanism for online identification of the pitch angle can be adopted to correct the normal ground view [1], as demonstrated in Fig. 7.4.

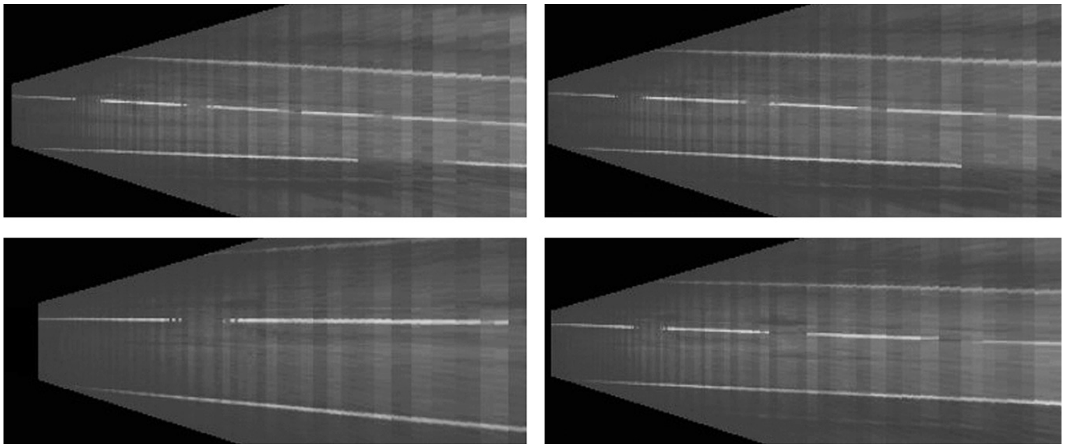


Fig. 7.4 Adaptive inverse perspective mapping [1]

In Sect. 4.5 in Chap. 4, why monocular vision is incapable of three-dimensional localization is clarified. This holds for general objects in the image. However, for ground objects, even monocular vision is enough to estimate their depths and reveal their three-dimensional world locations with respect to the intelligent vehicle. For an ground object, its $z_{\mathrm{w}} = 0$, substitute this equality into (7.1) and obtain

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{31} & m_{32} & m_{34} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix}.$$

Denote

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \equiv \begin{bmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{31} & m_{32} & m_{34} \end{bmatrix}^{-1}$$

and obtain

$$\frac{1}{\lambda} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{31} & m_{32} & m_{34} \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

which gives

$$x_{\mathrm{w}} = \frac{a_{11}u+a_{12}v+a_{13}}{a_{31}u+a_{32}v+a_{33}}, \tag{7.2a}$$

$$y_{\mathrm{w}} = \frac{a_{21}u+a_{22}v+a_{23}}{a_{31}u+a_{32}v+a_{33}}. \tag{7.2b}$$

For ground objects that are not completely in the ground surface plane, one can still compute their regions of interest reasonably via monocular vision. For example, suppose a pedestrian is standing at a place of image coordinates (u, v), then what is a proper region of interest for the pedestrian in the image?

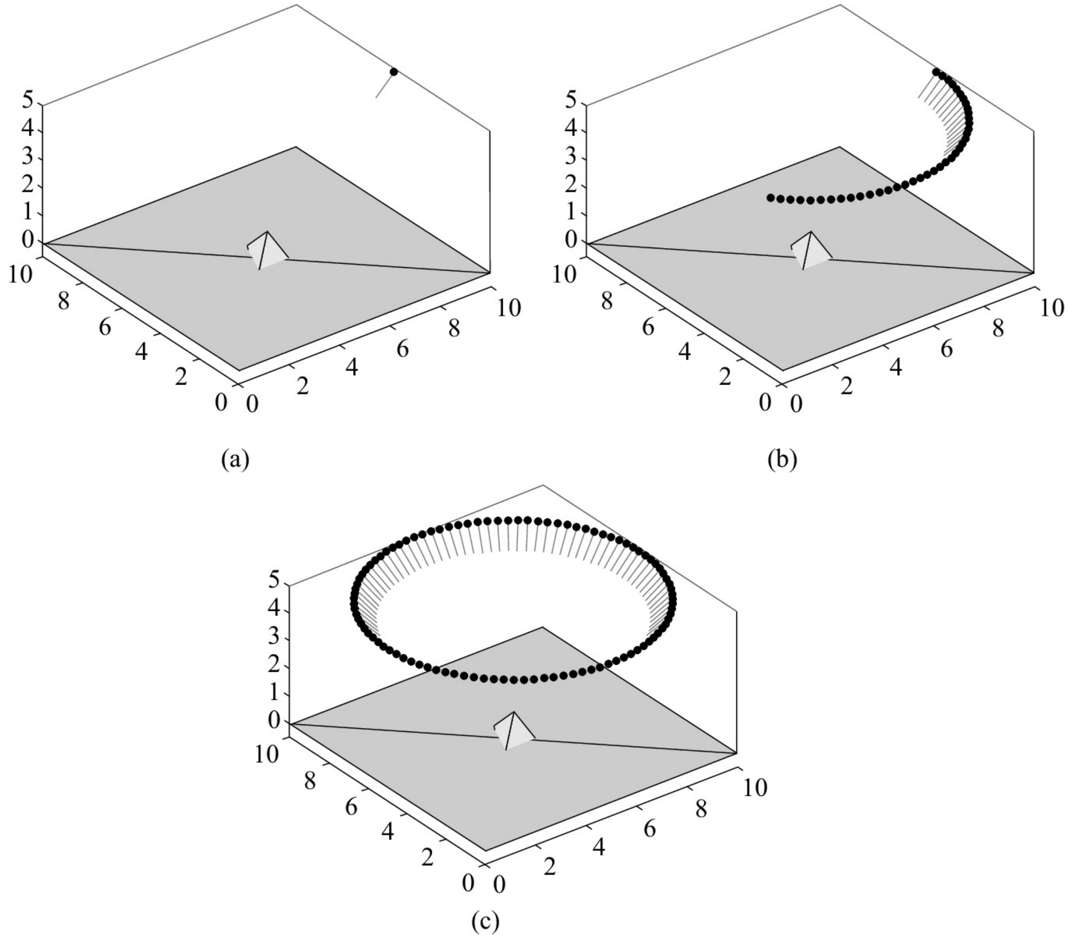
First, compute the standing place of the pedestrian in the vehicle-ground coordinates system via (7.2). Second, resort to the common sense that a pedestrian is usually not wider than a limit W (say one meter) and not higher than a limit H (say two meters), and set a bounding box with four corners

$$\begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} - \frac{W}{2} \\ 0 \end{bmatrix}, \quad \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} + \frac{W}{2} \\ 0 \end{bmatrix}, \quad \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} + \frac{W}{2} \\ H \end{bmatrix}, \quad \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} - \frac{W}{2} \\ H \end{bmatrix}$$

that contains the pedestrian. Third, project the four corner points onto the image via (7.1) to obtain a proper region of interest for the pedestrian.

7.2 Virtual Reality and Augmented Reality

The author presents a simple example to demonstrate how geometry fundamentals of computer vision can be applied in realization of virtual reality. Imagine a tetrahedron located in the center of a square carpet and a camera viewing around the tetrahedron, as illustrated in Fig. 7.5. More specifically, the camera starts from an initial pose as illustrated in Fig. 7.5a, and then rotates around the tetrahedron as illustrated in Fig. 7.5b, c. At each pose, the camera always views towards the tetrahedron. The intended task of virtual reality is to generate synthetic images as if captured by the camera during rotation around the tetrahedron.



(a) (b)

(c)

Fig. 7.5 A camera viewing around a tetrahedron in virtual reality **a** initial viewing pose; **b** an intermediate viewing pose; **c** final viewing pose

For the purpose, a feasible solution is as follows:

- Decompose surfaces of environment objects into triangles: From the perspective of the camera, there are five potentially visible triangles, namely the two triangles that form the carpet and the three triangles of the tetrahedron above the carpet, as illustrated in Fig. 7.5.
- For a generic pixel (u, v) of the synthetic image to be generated, compute potential intersections of its corresponding projective line with the triangles and keep the one closest to the camera.
- Set the color of the generic pixel (u, v) to the color of the closest intersection.

In above solution, the key point is how to compute a potential intersection of the projective line and a triangle. Given a generic triangle D-E-F, denote the positions of its three corners in the world coordinates system as $\boldsymbol{D}$, $\boldsymbol{E}$, and $\boldsymbol{F}$ respectively. First, derive the formalism of the projective line according to the pinhole camera model (4.7):

$$\begin{aligned} &\lambda \boldsymbol{P}_{\mathrm{i}}^{A} = \boldsymbol{A}[\boldsymbol{R}_{\mathrm{wc}} \quad \boldsymbol{T}_{\mathrm{wc}}]\boldsymbol{P}_{\mathrm{w}}^{A} \Longleftrightarrow \lambda \boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A} = \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{P}_{\mathrm{w}} + \boldsymbol{T}_{\mathrm{wc}} \\ \Leftrightarrow \quad &\boldsymbol{P}_{\mathrm{w}} = \boldsymbol{R}_{\mathrm{wc}}^{\mathrm{T}}(\lambda \boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A} - \boldsymbol{T}_{\mathrm{wc}}) = \lambda \boldsymbol{R}_{\mathrm{cw}}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A} + \boldsymbol{T}_{\mathrm{cw}} \end{aligned} \tag{7.3}$$

where $\boldsymbol{R}_{\mathrm{cw}} = \boldsymbol{R}_{\mathrm{wc}}^{\mathrm{T}}$ and $\boldsymbol{T}_{\mathrm{cw}} = -\boldsymbol{R}_{\mathrm{wc}}^{\mathrm{T}}\boldsymbol{T}_{\mathrm{wc}}$ denote the rotation and translation from the camera coordinates system to the world coordinates system.

Second, compute the intersection of the projective line of the pixel (u, v) and the plane spanned by the triangle D-E-F, which is represented by the equation

$$\boldsymbol{n}_{\Delta}^{\mathrm{T}}(\boldsymbol{P}_{\mathrm{w}} - \boldsymbol{D}) = 0 \tag{7.4}$$

where

$$\boldsymbol{n}_{\Delta} = \frac{(\boldsymbol{E}-\boldsymbol{D})\times(\boldsymbol{F}-\boldsymbol{D})}{\|(\boldsymbol{E}-\boldsymbol{D})\times(\boldsymbol{F}-\boldsymbol{D})\|_2}$$

denotes the normal vector of the plane spanned by the triangle D-E-F. Substitute (7.3) into (7.4) and solve λ that corresponds to the intersection point, namely

$$\boldsymbol{n}_{\Delta}^{\mathrm{T}}(\lambda \boldsymbol{R}_{\mathrm{cw}}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A} + \boldsymbol{T}_{\mathrm{cw}} - \boldsymbol{D}) = 0 \Longleftrightarrow \lambda = \frac{\boldsymbol{n}_{\Delta}^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{T}_{\mathrm{cw}})}{\boldsymbol{n}_{\Delta}^{\mathrm{T}}\boldsymbol{R}_{\mathrm{cw}}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A}}.$$

So the intersection point is

$$\boldsymbol{P}_{\mathrm{w}}^{I} = \frac{\boldsymbol{n}_{\Delta}^{\mathrm{T}}(\boldsymbol{D}-\boldsymbol{T}_{\mathrm{cw}})}{\boldsymbol{n}_{\Delta}^{\mathrm{T}}\boldsymbol{R}_{\mathrm{cw}}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A}}\boldsymbol{R}_{\mathrm{cw}}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{A} + \boldsymbol{T}_{\mathrm{cw}}. \tag{7.5}$$

Third, determine whether the intersection point described in (7.5) is inside the triangle D-E-F, using the method presented in Sect. 3.1.4 in Chap. 3. If the three points $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ do not belong to the degenerate case, then compute the coefficients of $\boldsymbol{P}_{\mathrm{w}}^{I}$ expressed linearly in terms of $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ by solving (3.20)

$$[\boldsymbol{D} \quad \boldsymbol{E} \quad \boldsymbol{F}]\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \boldsymbol{P}_{\mathrm{w}}^{I}$$

and check if the non-negative condition (3.22), i.e.

$$0 \le t_1 \le 1, \quad 0 \le t_2 \le 1, \quad 0 \le t_3 \le 1$$

is satisfied or not. If the condition (3.22) is satisfied, $\boldsymbol{P}_{\mathrm{w}}^{I}$ is inside the triangle *D*-*E*-*F*; otherwise, not inside. It is worth noting that since $\boldsymbol{P}_{\mathrm{w}}^{I}$ is located on the plane spanned by the triangle *D*-*E*-*F*, the normalization condition (3.21), i.e.

$$t_1 + t_2 + t_3 = 1$$

is naturally satisfied.

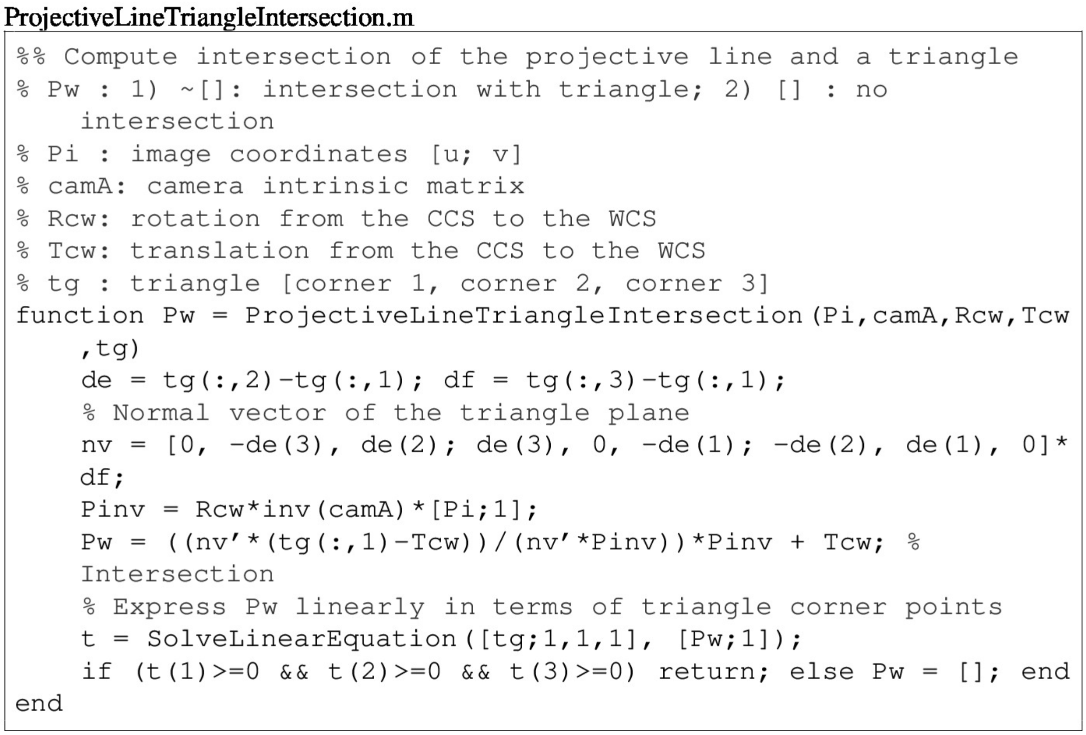
If the three points $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ belong to the degenerate case (i.e. the origin $\boldsymbol{O}$ is located on the plane spanned by the triangle *D*-*E*-*F*), then compute the coefficients t_1, t_2, t_3 by solving (3.23), i.e. the augmented version of (3.20)

$$\begin{bmatrix} \boldsymbol{D} & \boldsymbol{E} & \boldsymbol{F} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_{\mathrm{w}}^{I} \\ 1 \end{bmatrix}$$

and then check if the non-negative condition (3.22) is satisfied or not. In fact, be the three points $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ belonging to the degenerate case or not, the coefficients t_1, t_2, t_3 can always be computed by solving (3.23): If the three points $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ do not belong to the degenerate case, then as explained above, $\boldsymbol{P}_{\mathrm{w}}^{I}$ is located on the plane spanned by the triangle *D*-*E*-*F*, so the normalization condition (3.21), which is right the augmented part, i.e. the last equation of (3.23), is just a trivial constraint that can be naturally satisfied by the solution of (3.23). If the three points $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$ belong to the degenerate case, then the augmented equation of (3.23) can resolve ambiguity of (3.20) and force uniqueness of the solution that satisfies (3.20).

Finally, just discard the intersection point described in (7.5) if it is not inside the triangle *D*-*E*-*F*, because in such case, the projective line of the pixel (u, v) does not actually intersect with the triangle *D*-*E*-*F*.

Matlab code for computing the potential intersection of the projective line of a given pixel and a given triangle is given as follows.


ProjectiveLineTriangleIntersection.m

```
%% Compute intersection of the projective line and a triangle
% Pw : 1) ~[]: intersection with triangle; 2) [] : no
    intersection
% Pi : image coordinates [u; v]
% camA: camera intrinsic matrix
% Rcw: rotation from the CCS to the WCS
% Tcw: translation from the CCS to the WCS
% tg : triangle [corner 1, corner 2, corner 3]
function Pw = ProjectiveLineTriangleIntersection(Pi,camA,Rcw,Tcw
    ,tg)
    de = tg(:,2)-tg(:,1); df = tg(:,3)-tg(:,1);
    % Normal vector of the triangle plane
    nv = [0, -de(3), de(2); de(3), 0, -de(1); -de(2), de(1), 0]*
    df;
    Pinv = Rcw*inv(camA)*[Pi;1];
    Pw = ((nv'*(tg(:,1)-Tcw))/(nv'*Pinv))*Pinv + Tcw; %
    Intersection
    % Express Pw linearly in terms of triangle corner points
    t = SolveLinearEquation([tg;1,1,1], [Pw;1]);
    if (t(1)>=0 && t(2)>=0 && t(3)>=0) return; else Pw = []; end
end
```

Matlab code for demonstrating the animated effect of virtual reality from the perspective of a camera viewing around a tetrahedron is given as follows. Generated synthetic images for realizing virtual reality are demonstrated in Fig. 7.6.

demoTetrahedronVR.m

```
%% Demonstrate animated virtual reality of a tetrahedron
sz=10; % Carpet width
C1=[0;0;0]; C2=[sz;0;0]; C3=[sz;sz;0]; C4=[0;sz;0]; % Carpet
    corners
% D-E-F-G : Tetrahedron corners
a=1; D=[sz/2;sz/2;sqrt(2)*a]; E=[sz/2+a;sz/2;0];
F=[sz/2-a/2;sz/2+sqrt(3/4)*a;0]; G=[sz/2-a/2;sz/2-sqrt(3/4)*a
    ;0];
tg = zeros(3,3,5); % Triangles
tg(:,:,1)=[C1,C2,C4]; tg(:,:,2)=[C2,C3,C4]; % On carpet
tg(:,:,3)=[D,E,F]; tg(:,:,4)=[D,F,G]; tg(:,:,5)=[D,G,E]; % On
    tetrahedron
tgc = [200,200,200; 200,200,200; 255,0,0; 0,255,0; 0,0,255]; %
    Colors

alf = 400; u0 = 320; v0 = 240; W = 2*u0; H = 2*v0;
A = [alf, 0, u0; 0, alf, v0; 0, 0, 1]; % Camera intrinsic matrix
imgS = uint8(zeros(H,W,3)); % Set the demonstration image
cr = sz/2; % Camera rotating radius
ch = 5; % Camera height
Rcw = zeros(3,3); % Rotation from CCS to WCS
a1 = pi/4; % Camera Euler angle-1
idx = 0; % Synthetic image index
for a2 = 2*pi:-pi/45:pi/45 % Camera Euler angle-2
    imgS(:,:,:) = 0; % Reset the demonstration image
    Tcw = [cr+cr*cos(a2);cr+cr*sin(a2);ch]; % Translation from
    CCS to WCS
    Rcw(:,1) = [sin(a2);-cos(a2);0];
    Rcw(:,2) = [-sin(a1)*cos(a2);-sin(a1)*sin(a2);cos(a1)];
    Rcw(:,3) = [-cos(a1)*cos(a2);-cos(a1)*sin(a2);-sin(a1)];
    for v=1:H
        for u=1:W
            dm = 10000;
            for k=1:size(tg,3)
                Pw = ProjectiveLineTriangleIntersection(...
                    [u;v], A, Rcw, Tcw, tg(:,:,k));
                if (~ isempty(Pw))
                    dtmp = (Pw-Tcw)’*(Pw-Tcw);
                    if (dtmp<dm) dm = dtmp; imgS(v,u,:) = tgc(k
    ,:); end
                end
            end
        end
    end
    figure(1), imgS = imgS(end:-1:1,end:-1:1,:); imshow(imgS);
    pause(0.1);
    idx=idx+1; imwrite(imgS,[’tetrahedron\view\_’ num2str(idx) ’
    .jpg’]);
end
```

Geometry fundamentals of computer vision can also be applied in realization of augmented reality. It is worth distinguishing between virtual reality and augmented reality. The former refers to generating completely synthetic images that simulate the real world as well as purely imagined reality, whereas the latter refers to augmenting images actually captured from the real world by virtual elements to facilitate conception of the images.

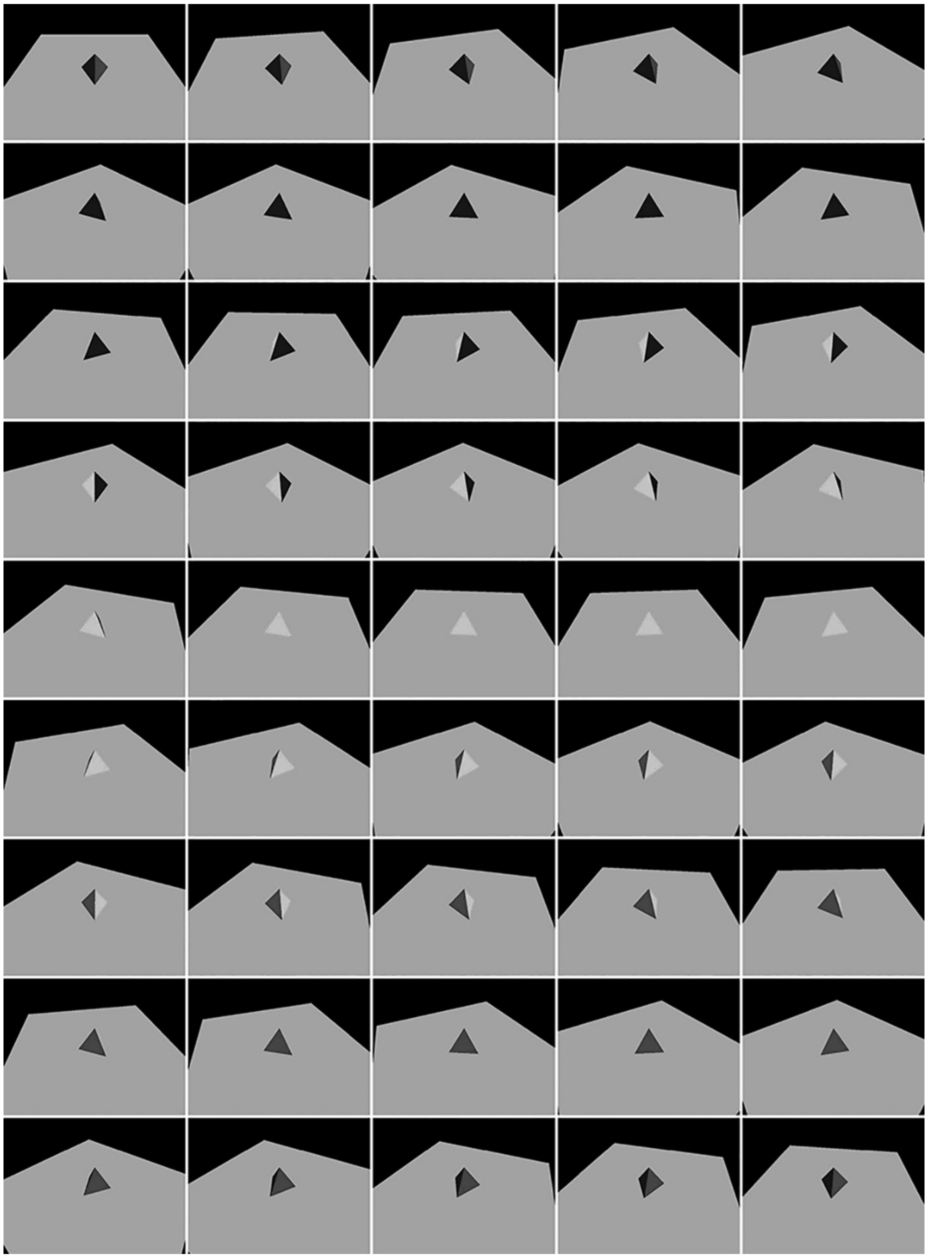


Fig. 7.6 Virtual reality from the perspective of a camera viewing around a tetrahedron

An example of augmented reality has already been demonstrated in Fig. 5.6, where the visualized laser points and their ground projections are not actually captured by the camera but are virtual elements that augment the original image and facilitate one's conception of which environment parts are scanned by the laser scanner. In fact, the spirit of augmented reality is reflected in many figures in previous Chapters. For more examples, in Fig. 2.27, the original image is augmented by visualized SIFT features. In Figs. 4.17 and 4.18, the original and rectified images are augmented by visualized epipolar lines.

7.3 Cooperative Augmented Reality: See-Through Effect

Geometry fundamentals of computer vision can be applied to cooperative intelligent systems to form a special kind of augmented reality coined as **cooperative augmented reality** [3–5]. Consider a traffic scenario as illustrated in Fig. 7.7, where the first intelligent vehicle occludes the view of the second intelligent vehicle. From the perspective of the second intelligent vehicle, it would always be desirable to have certain inference of environment objects occluded by the first intelligent vehicle. However, this cannot be achieved by the second intelligent vehicle alone.

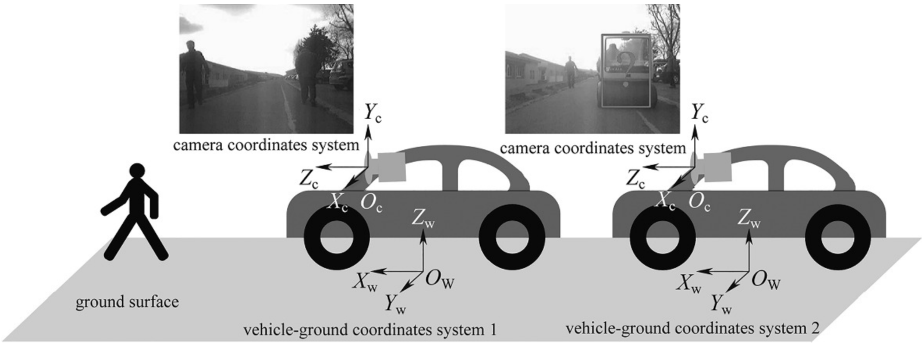

Y_c
camera coordinates system
Z_c
X_c O_c
Z_w
X_w O_W
Y_w
Y_c
camera coordinates system
Z_c
X_c O_c
Z_w
X_w O_W
Y_w
ground surface
vehicle-ground coordinates system 1
vehicle-ground coordinates system 2

Fig. 7.7 Cooperative intelligent vehicles

For purpose of enabling the second intelligent vehicle to somehow “perceive” environment objects occluded by the first intelligent vehicle, one may resort to the methodology of **cooperative perception** [4] between the first intelligent vehicle and the second intelligent vehicle. More specifically, the first intelligent vehicle shares its perception with the second intelligent vehicle via inter-vehicle communication,[1] then the second intelligent vehicle associates its own perception with that shared by the first intelligent vehicle, and establishes the relationship between the image coordinates system attached to itself and that attached to the first intelligent vehicle.

Suppose for the first intelligent vehicle, the perspective mapping between the vehicle-ground coordinates system and the image coordinates system is described by

$$\lambda \begin{bmatrix} u^{[1]} \\ v^{[1]} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}^{[1]} & m_{12}^{[1]} & m_{13}^{[1]} & m_{14}^{[1]} \\ \text{[6pt]}m_{21}^{[1]} & m_{22}^{[1]} & m_{23}^{[1]} & m_{24}^{[1]} \\ \text{[6pt]}m_{31}^{[1]} & m_{32}^{[1]} & m_{33}^{[1]} & m_{34}^{[1]} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}}^{[1]} \\ y_{\mathrm{w}}^{[1]} \\ z_{\mathrm{w}}^{[1]} \\ 1 \end{bmatrix}$$

and for the second intelligent vehicle, the perspective mapping between the vehicle-ground coordinates system and the image coordinates system is described by

$$\lambda\begin{bmatrix}u^{[2]}\\ v^{[2]}\\ 1\end{bmatrix}=\begin{bmatrix}m_{11}^{[2]} & m_{12}^{[2]} & m_{13}^{[2]} & m_{14}^{[2]}\\ [6pt]m_{21}^{[2]} & m_{22}^{[2]} & m_{23}^{[2]} & m_{24}^{[2]}\\ [6pt]m_{31}^{[2]} & m_{32}^{[2]} & m_{33}^{[2]} & m_{34}^{[2]}\end{bmatrix}\begin{bmatrix}x_{\mathrm{w}}^{[2]}\\ y_{\mathrm{w}}^{[2]}\\ z_{\mathrm{w}}^{[2]}\\ 1\end{bmatrix}.$$

Further suppose the inter-vehicle relative pose [6, 7] estimated via perception association is represented by the rotation matrix $\boldsymbol{R}_{12}$ and the translation vector $\boldsymbol{T}_{12}$. Besides, suppose depths, i.e. x_{w} coordinates of environment objects can be estimated by the first intelligent vehicle. Visualization of estimated depths of environment objects in an example scenario is demonstrated in Fig. 7.8.

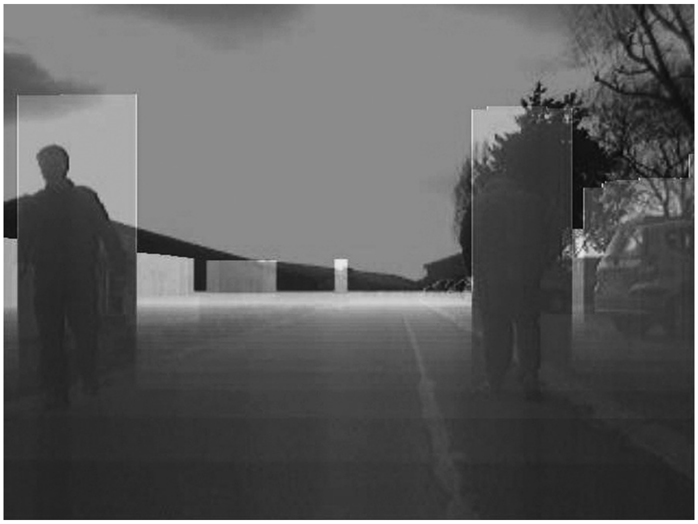


Fig. 7.8 Visualization of estimated depths of environment objects [4]

Given a generic pixel $\left(u^{[1]}, v^{[1]}\right)$ in the image coordinates system of the first intelligent vehicle, since its corresponding depth $x_{\mathrm{w}}^{[1]}$ is assumed available, the pixel’s counterpart in the vehicle-ground coordinates system of the first intelligent vehicle is computed via

$$\begin{aligned}
&\lambda\begin{bmatrix}u^{[1]}\\ v^{[1]}\\ 1\end{bmatrix}=\begin{bmatrix}m_{11}^{[1]} & m_{12}^{[1]} & m_{13}^{[1]} & m_{14}^{[1]}\\ m_{21}^{[1]} & m_{22}^{[1]} & m_{23}^{[1]} & m_{24}^{[1]}\\ m_{31}^{[1]} & m_{32}^{[1]} & m_{33}^{[1]} & m_{34}^{[1]}\end{bmatrix}\begin{bmatrix}x_{\mathrm{w}}^{[1]}\\ y_{\mathrm{w}}^{[1]}\\ z_{\mathrm{w}}^{[1]}\\ 1\end{bmatrix}\\
\Leftrightarrow\quad & x_{\mathrm{w}}^{[1]}\begin{bmatrix}m_{11}^{[1]}\\ m_{21}^{[1]}\\ m_{31}^{[1]}\end{bmatrix}+y_{\mathrm{w}}^{[1]}\begin{bmatrix}m_{12}^{[1]}\\ m_{22}^{[1]}\\ m_{32}^{[1]}\end{bmatrix}+z_{\mathrm{w}}^{[1]}\begin{bmatrix}m_{13}^{[1]}\\ m_{23}^{[1]}\\ m_{33}^{[1]}\end{bmatrix}+\begin{bmatrix}m_{14}^{[1]}\\ m_{24}^{[1]}\\ m_{34}^{[1]}\end{bmatrix}-\lambda\begin{bmatrix}u^{[1]}\\ v^{[1]}\\ 1\end{bmatrix}=\mathbf{0}\\
\Leftrightarrow\quad & \begin{bmatrix}m_{12}^{[1]} & m_{13}^{[1]} & u^{[1]}\\ m_{22}^{[1]} & m_{23}^{[1]} & v^{[1]}\\ m_{32}^{[1]} & m_{33}^{[1]} & 1\end{bmatrix}\begin{bmatrix}y_{\mathrm{w}}^{[1]}\\ z_{\mathrm{w}}^{[1]}\\ -\lambda\end{bmatrix}=-(x_{\mathrm{w}}^{[1]}\begin{bmatrix}m_{11}^{[1]}\\ m_{21}^{[1]}\\ m_{31}^{[1]}\end{bmatrix}+\begin{bmatrix}m_{14}^{[1]}\\ m_{24}^{[1]}\\ m_{34}^{[1]}\end{bmatrix})\\
\Leftrightarrow\quad & \begin{bmatrix}y_{\mathrm{w}}^{[1]}\\ z_{\mathrm{w}}^{[1]}\\ -\lambda\end{bmatrix}=-\begin{bmatrix}m_{12}^{[1]} & m_{13}^{[1]} & u^{[1]}\\ m_{22}^{[1]} & m_{23}^{[1]} & v^{[1]}\\ m_{32}^{[1]} & m_{33}^{[1]} & 1\end{bmatrix}^{-1}\begin{bmatrix}x_{\mathrm{w}}^{[1]}m_{11}^{[1]}+m_{14}^{[1]}\\ x_{\mathrm{w}}^{[1]}m_{21}^{[1]}+m_{24}^{[1]}\\ x_{\mathrm{w}}^{[1]}m_{31}^{[1]}+m_{34}^{[1]}\end{bmatrix}.
\end{aligned}\tag{7.6}$$

Then transform the world coordinates expressed in the vehicle-ground coordinates system of the first intelligent vehicle to those expressed in vehicle-ground coordinates system of the second intelligent vehicle via

$$\begin{bmatrix}x_{\mathrm{w}}^{[2]}\\ y_{\mathrm{w}}^{[2]}\\ z_{\mathrm{w}}^{[2]}\\ 1\end{bmatrix}=\begin{bmatrix}\boldsymbol{R}_{12} & \boldsymbol{T}_{12}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}\begin{bmatrix}x_{\mathrm{w}}^{[1]}\\ y_{\mathrm{w}}^{[1]}\\ z_{\mathrm{w}}^{[1]}\\ 1\end{bmatrix}.\tag{7.7}$$

Substitute (7.7) into the perspective mapping associated with the second intelligent vehicle and obtain

$$\lambda\begin{bmatrix}u^{[2]}\\ v^{[2]}\\ 1\end{bmatrix}=\begin{bmatrix}m_{11}^{[2]} & m_{12}^{[2]} & m_{13}^{[2]} & m_{14}^{[2]}\\ m_{21}^{[2]} & m_{22}^{[2]} & m_{23}^{[2]} & m_{24}^{[2]}\\ m_{31}^{[2]} & m_{32}^{[2]} & m_{33}^{[2]} & m_{34}^{[2]}\end{bmatrix}\begin{bmatrix}\boldsymbol{R}_{12} & \boldsymbol{T}_{12}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}\begin{bmatrix}x_{\mathrm{w}}^{[1]}\\ y_{\mathrm{w}}^{[1]}\\ z_{\mathrm{w}}^{[1]}\\ 1\end{bmatrix}.\tag{7.8}$$

Finally substitute (7.6) into (7.8) and obtain

$$\lambda \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}^{[2]} & m_{12}^{[2]} & m_{13}^{[2]} & m_{14}^{[2]} \\ m_{21}^{[2]} & m_{22}^{[2]} & m_{23}^{[2]} & m_{24}^{[2]} \\ m_{31}^{[2]} & m_{32}^{[2]} & m_{33}^{[2]} & m_{34}^{[2]} \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{12} & \boldsymbol{T}_{12} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \left(\begin{bmatrix} x_{\mathrm{w}}^{[1]} \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} m_{12}^{[1]} & m_{13}^{[1]} & u^{[1]} \\ m_{22}^{[1]} & m_{23}^{[1]} & v^{[1]} \\ m_{32}^{[1]} & m_{33}^{[1]} & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_w^{[1]} m_{11}^{[1]} + m_{14}^{[1]} \\ x_w^{[1]} m_{21}^{[1]} + m_{24}^{[1]} \\ x_w^{[1]} m_{31}^{[1]} + m_{34}^{[1]} \end{bmatrix} \right) \tag{7.9}$$

which formalizes the relationship between the image coordinates systems attached to the two intelligent vehicles.

Thanks to (7.9), the second intelligent vehicle can take advantage of perception shared by the first intelligent vehicle and project pixels of occluded environment objects onto its own image to form the cooperative augmented reality effect of seeing-through another vehicle, as demonstrated in Fig. 7.9.

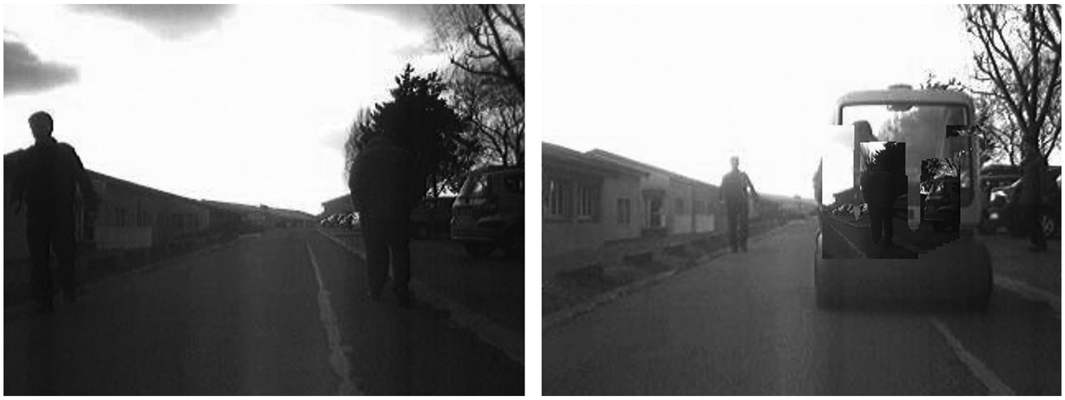


Fig. 7.9 Cooperative augmented reality: see-through from the camera perspective [4]

Besides the cooperative augmented reality effect of seeing-through from the camera perspective, one can also follow the same methodology to form the cooperative augmented reality effect of seeing-through from the driver perspective, as demonstrated in Fig. 7.10.

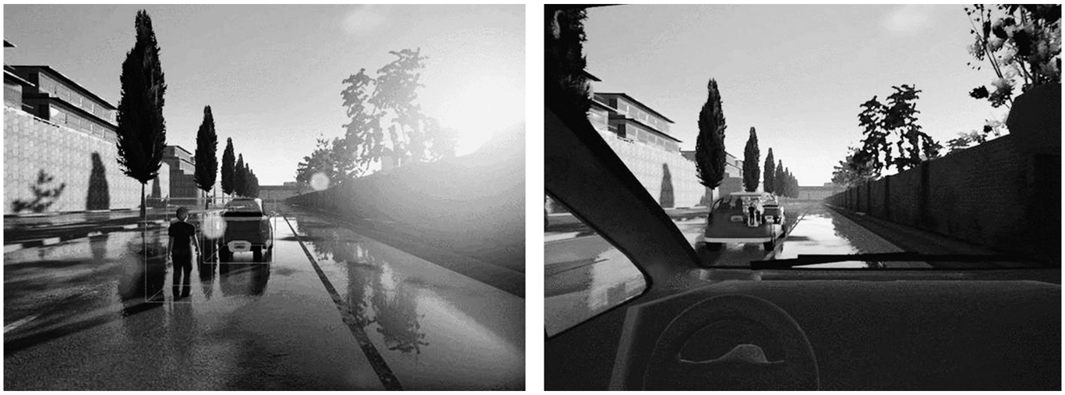


Fig. 7.10 Cooperative augmented reality: see-through from the driver perspective

7.4 Image Stitching

Another elegant application of geometry fundamentals of computer vision is **image stitching**, which refers to the practice of stitching multiple images that capture various views of the environment into a larger merged image that gives a panorama of the environment. For example, given a number of original images illustrated in Fig. 7.11, which are captured from the same location at Huang-Shan, i.e. Yellow-Mountain but with viewing angle differences. How to stitch the images into one image is demonstrated below.

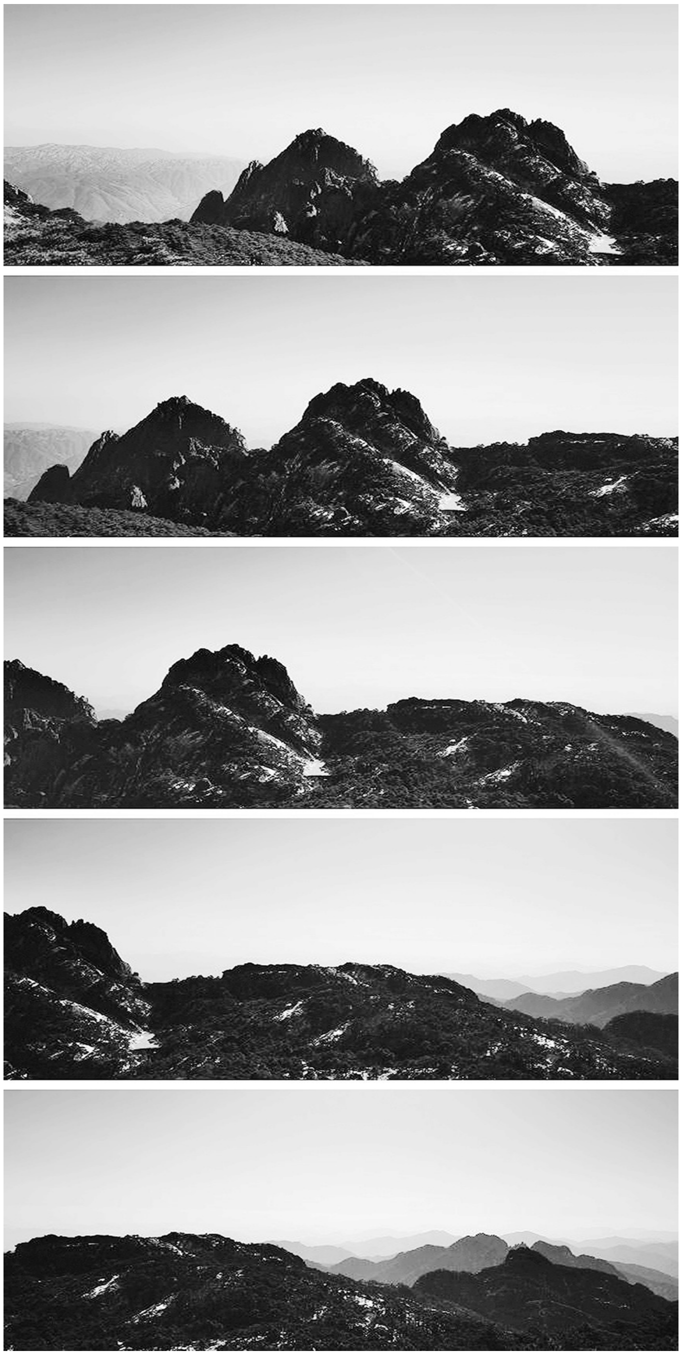


Fig. 7.11 Original images to be stitched

Note that if one knows how to stitch two images, one can naturally know how to stitch arbitrarily multiple images. It is like if one knows how to add two numbers, one can know how to add arbitrarily multiple numbers. So consider stitching of two generic images that have enough view part in common. Set the world coordinates system identical to the first camera coordinates system, i.e. the coordinates system of the camera when capturing the first image. Formalize the relationship between the world coordinates system and the first image coordinates system as

$$\lambda_1 \begin{bmatrix} u^{[1]} \\ v^{[1]} \\ 1 \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} x_{\mathrm{c}}^{[1]} \\ y_{\mathrm{c}}^{[1]} \\ z_{\mathrm{c}}^{[1]} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \end{bmatrix} \tag{7.10}$$

and formalize the relationship between the world coordinates system and the second image coordinates system as

$$\lambda_2 \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} x_{\mathrm{c}}^{[2]} \\ y_{\mathrm{c}}^{[2]} \\ z_{\mathrm{c}}^{[2]} \end{bmatrix} = \boldsymbol{A} \left(\boldsymbol{R}_{12} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \end{bmatrix} + \boldsymbol{T}_{12} \right), \tag{7.11}$$

where $\boldsymbol{R}_{12}$ and $\boldsymbol{T}_{12}$ denote the rotation matrix and the translation vector from the first camera coordinates system to the second camera coordinates system.

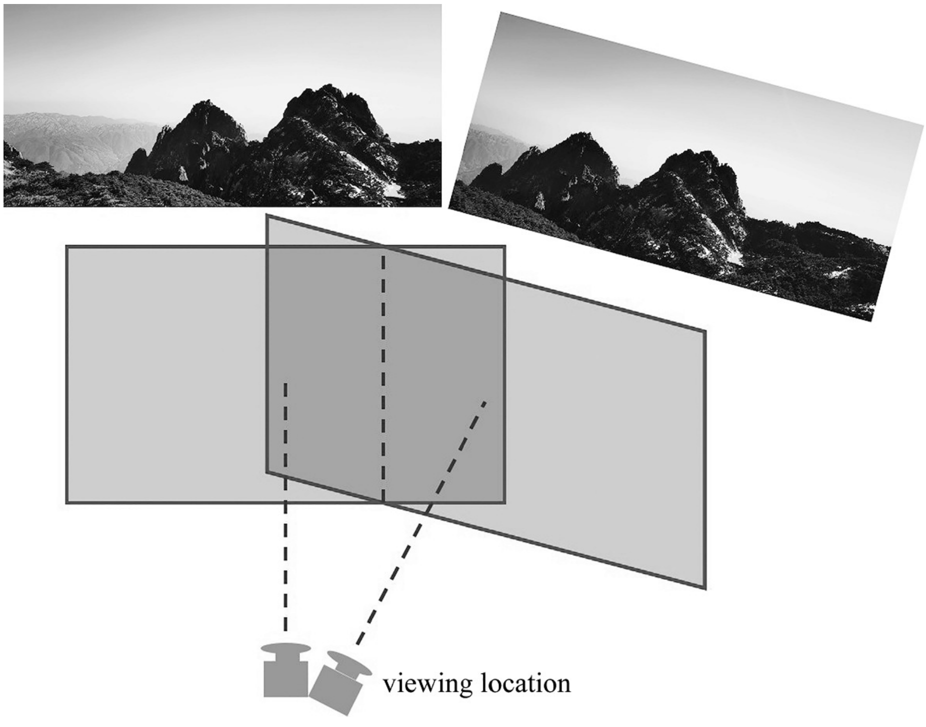


Fig. 7.12 Homography approximated relationship between images

If the distances of environment objects to the two camera coordinates systems are much larger than the distance between the two camera coordinates systems, i.e. if

$$\left\| \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \end{bmatrix} \right\|_2 \gg \|\boldsymbol{T}_{12}\|_2,$$

which is the case for many practical applications that involve image stitching, then (7.11) can be fairly approximated by

$$\lambda_2 \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} \approx \boldsymbol{A}\boldsymbol{R}_{12} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \end{bmatrix}. \tag{7.12}$$

Substitute (7.12) into (7.10) and obtain

$$\lambda_2(\boldsymbol{A}\boldsymbol{R}_{12})^{-1} \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} \approx \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ z_{\mathrm{w}} \end{bmatrix} = \lambda_1 \boldsymbol{A}^{-1} \begin{bmatrix} u^{[1]} \\ v^{[1]} \\ 1 \end{bmatrix} \Longleftrightarrow \lambda_{12} \begin{bmatrix} u^{[2]} \\ v^{[2]} \\ 1 \end{bmatrix} \approx \boldsymbol{H}_{12} \begin{bmatrix} u^{[1]} \\ v^{[1]} \\ 1 \end{bmatrix} \tag{7.13}$$

where

$$\lambda_{12} \equiv \lambda_2/\lambda_1, \quad \boldsymbol{H}_{12} \equiv \boldsymbol{A}\boldsymbol{R}_{12}\boldsymbol{A}^{-1}.$$

In other words, the relationship between the two image coordinates systems can be fairly approximated by homography, as illustrated in Fig. 7.12.

In practice, one needs to realize real-time calibration or estimation of the approximated homography between the two images. For the purpose, one can take advantage of certain advanced features such as SIFT features and ORB features presented in Chap. 2. Extract features from both images, and then match features between the two images according to feature similarities. Matlab code for matching SIFT feature points between a pair of images is given as follows.

MatchSIFT.m

```
%% Match SIFT Points
% fD: SIFT points’ corresponding (vector) descriptors
% iP: Indices and distances of pairs between two sets of SIFT
    descriptors
%    row 1: Indices of matched SIFT points in the first set
%    row 2: Indices of matched SIFT points in the second set
%    row 3: Feature-space distances between matched SIFT points
    in pairs
function iP = MatchSIFT(fDa, fDb, tDist)
    % tDist: Distance threshold for each pair of matched SIFT
    points
    if (nargin<3) tDist = 0.06; end
    na = size(fDa,2); nb = size(fDb,2);
    iPa = zeros(2,na); iPb = zeros(2,nb); iP = zeros(3,na);
    for k = 1:na
        d = fDa(:,k)*ones(1,nb)-fDb; d = sum(d.^2,1); dMin = min
    (d);
        iMin = find(dMin==d); iMin = iMin(1); iPa(:,k) = [iMin;
    dMin];
    end
    for k = 1:nb
        d = fDb(:,k)*ones(1,na)-fDa; d = sum(d.^2,1); dMin = min
    (d);
        iMin = find(dMin==d); iMin = iMin(1); iPb(:,k) = [iMin;
    dMin];
    end
    for ia = 1:na
        ib = iPa(1,ia);
        if (iPb(1,ib)==ia) % Matching is mutually consistent
            iP(:,ia) = [ia; ib; sqrt(iPa(2,ia))];
        end
    end
    k = find(iP(1,:)>0 & iP(3,:)<tDist); iP = iP(:,k);
end
```

Matlab code for matching ORB feature points between a pair of images is given as follows. The demonstrated Matlab code takes advantage of *bit operations* to enhance computational efficiency. More specifically, each 256-dimensional feature descriptor vector is stored in four 64-bit integers. The bit-wise *XOR* (*eXclusive OR*) operation is used to compute the *Hamming distance* [8] that characterizes the difference between two feature descriptors.

MatchORB.m

```
%% Match ORB Points
% fD: ORB points’ corresponding descriptors
% iP: Indices and distances of matched ORB pairs
%    row 1: Indices of matched ORB points in the first set
%    row 2: Indices of matched ORB points in the second set
%    row 3: Hamming distances between matched ORB pairs
function iP = MatchORB(fDa, fDb, tDist)
    % tDist: Distance threshold for matched ORB pairs
    if (nargin<3) tDist = 25; end
    na = size(fDa,2); nb = size(fDb,2);
    iPa = zeros(2,na); iPb = zeros(2,nb); iP = zeros(3,na);
    for k = 1:na
        % Take advantage of bit-XOR for fast computation
        d=BitSum(bitxor(repmat(fDa(:,k),1,nb),fDb)); dM=min(d);
        iM = find(dM==d); iM = iM(1); iPa(:,k) = [iM; dM];
    end
    for k = 1:nb
        % Take advantage of bit-XOR for fast computation
        d=BitSum(bitxor(repmat(fDb(:,k),1,na),fDa)); dM=min(d);
        iM = find(dM==d); iM = iM(1); iPb(:,k) = [iM; dM];
    end
    for ia = 1:na
        ib = iPa(1,ia);
        if (iPb(1,ib)==ia) % Matching is mutually consistent
            iP(:,ia) = [ia; ib; iPa(2,ia)];
        end
    end
    k = find(iP(1,:)>0 & iP(3,:)<tDist); iP = iP(:,k);

    %% Summate bits along each column
    function B = BitSum(M)
        B = zeros(1,size(M,2));
        for j=1:64
            B = B + sum(bitget(M,j),1);
        end
    end
end
```

Once feature matching between the two images is done and if the established correspondences among feature points are assumed all correct, then the remaining step is just to compute the homography matrix between the two images via the method presented in Sect. 4.3 in Chap. 4. In practice, however, the established correspondences among feature points tend to have outliers, i.e. wrong ones. If all correspondences of feature points are used to compute the homography matrix, those outlier or wrong correspondences of feature points are likely to cause the computed result to deteriorate severely.

If there was *a priori* knowledge of which correspondences of feature points are correct whereas which ones are wrong, the homography matrix would still be computed effectively, only if wrong correspondences of feature points are excluded from computation. However, there is normally no such *a priori* knowledge in practice.

To handle the problem, one can resort to the RANdom SAmple Consensus (RANSAC) method presented in Sect. 6.3.3 in Chap. 6. The concrete RANSAC method for computing the homography matrix is as follows:

- Randomly select four correspondences of feature points (be them SIFT features, ORB features, or others) to compute a homography matrix.
- Evaluate the number of all correspondences of feature points that are consistent with the homography matrix fitted by the four sample correspondences of feature points.
- Repeat above sampling and evaluation process.
- Finally take the best homography matrix namely the one that is consistent with the largest number of correspondences of feature points. Collect all correspondences of feature points that are consistent with the best homography matrix and discard all those not so. Use all consistent correspondences of feature points to compute a new homography matrix and output it as the final result.

Matlab code for computing the homography matrix via the RANSAC method is given as follows. The involved sub-functions **CalibrateHomography** and **HomographyXY2UV** have already been given in Chap. 4.

MaxHomographyRANSAC.m

```
%% Maximize homography consensus via RANSAC
% Compute homography maximizing correspondence consensus
% i.e. consistent pairs between two images
% pairPt:
%    row 1: u-coordinates of matched features in image1
%    row 2: v-coordinates of matched features in image1
%    row 3: u-coordinates of matched features in image2
%    row 4: v-coordinates of matched features in image2
% Hm: Homography from image1 to image2
function Hm = MaxHomographyRANSAC(pairPt)
    etol = 5; cn = 0; ransacN = 100;
    u1 = pairPt(1,:); v1 = pairPt(2,:);
    u2 = pairPt(3,:); v2 = pairPt(4,:);
    for k = 1:ransacN
        % Random sampling
        i = ceil(rand(1,4)*size(pairPt,2));
        u1t=u1(i); v1t=v1(i); u2t=u2(i); v2t=v2(i);
        Hr = CalibrateHomography(u2t, v2t, u1t, v1t);
        [u2p,v2p]=HomographyXY2UV(Hr,u1,v1);
        % Consensus
        pdif=abs(u2p-u2)+abs(v2p-v2);i=find(pdif<etol);
        u1t=u1(i); v1t=v1(i); u2t=u2(i); v2t=v2(i);
        i = length(i);
        if (i>cn)
            fprintf('%d-th RANSAC: %d good pairs\n',k,i);
            cn = i;
            Hm = CalibrateHomography(u2t,v2t,u1t,v1t);
        end
    end
    fprintf('FINAL (N=%d): %d good pairs \n',ransacN,cn);
end
```

Once the homography between the two images is known, then they can be stitched into one image. For arbitrarily multiple images, their relevant homography matrices can be computed one by one, and they can be stitched into the final image one by one. Matlab code for image stitching is given as follows.

StitchImages.m

```
%% Stitch images indexed from left to right
function [imgF, f] = StitchImages(f, ftname)
    for k = 1:length(f)
        f(k).img = rgb2gray(f(k).imgc);
        f(k).H=size(f(k).img,1); f(k).W=size(f(k).img,2);
        if (isequal(ftname,’SIFT’))
            [fL, fD, oG, oDG] = SIFT(f(k).img);
            fL = fL(5:6,:);
        elseif (isequal(ftname,’ORB’))
            [fL, fD] = ORB(f(k).img);
        end
        f(k).fL = fL; f(k).fD = fD;
    end
    for k = 1:length(f)−1
        if (isequal(ftname,’SIFT’))
            iP = MatchSIFT(f(k).fD,f(k+1).fD);
        elseif (isequal(ftname,’ORB’))
            iP = MatchORB(f(k).fD,f(k+1).fD);
        end
        pairPt = [f(k).fL(1,iP(1,:)); ...
                  f(k).fL(2,iP(1,:)); ...
                  f(k+1).fL(1,iP(2,:)); ...
                  f(k+1).fL(2,iP(2,:))];
        Hm = MaxHomographyRANSAC(pairPt);
        f(k+1).Hleft = Hm; % Homography from left
        f(k).Hright = inv(Hm); % Homography from right
    end
    m = floor((length(f)+1)/2); f(m).Hm = eye(3);
    umin = 1; umax = f(m).W; vmin = 1; vmax = f(m).H;
    for k = 1:length(f)
        Hm = f(m).Hm; H = f(k).H; W = f(k).W;
        if (k<m)
            for j=m−1:−1:k Hm = f(j).Hright*Hm; end
        elseif (k>m)
            for j=m+1:k Hm = f(j).Hleft*Hm; end
        else continue; end
        f(k).Hm = Hm; iHm = inv(Hm);
        [ut,vt]=HomographyXY2UV(iHm,[1,1,W,W],[1,H,1,H]);
        f(k).umin=floor(min(ut)); f(k).umax=ceil(max(ut));
        f(k).vmin=floor(min(vt)); f(k).vmax=ceil(max(vt));
        umin=min(umin,f(k).umin); umax=max(umax,f(k).umax);
        vmin=min(vmin,f(k).vmin); vmax=max(vmax,f(k).vmax);
    end
    imgF = uint8(zeros(vmax−vmin+1,umax−umin+1,3));
    du = 1−umin; dv = 1−vmin; imgF(:,:,:) = 255;
    for k=[length(f):−1:m+1 1:m−1]
        Hm = f(k).Hm; H = f(k).H; W = f(k).W;
        for v=f(k).vmin:f(k).vmax
            for u=f(k).umin:f(k).umax
                ut = u+du; vt = v+dv;
                [up,vp] = HomographyXY2UV(Hm,u,v);
                up = round(up); vp = round(vp);
                if (up>=1 && up<=W && vp>=1 && vp<=H)
                    imgF(vt,ut,:) = f(k).imgc(vp,up,:);
                end
            end
        end
    end
    imgF(1+dv:f(m).H+dv,1+du:f(m).W+du,:) = f(m).imgc;
end
```

Given the original images illustrated in Fig. 7.11, the effect of stitching them based on SIFT features is demonstrated in Fig. 7.13, whereas the effect of stitching them based on ORB features is demonstrated in Fig. 7.14. It is worth noting that in the stitched image, slight color inconsistency along original image boundaries is intentionally kept to let readers see the “influencing” areas of the original images in the stitched image. Such slight color inconsistency itself can be easily handled in practice.

To generate a stitched image demonstrated in Fig. 7.13 or Fig. 7.14, one usually needs to select a baseline image plane and then to project the original images onto the baseline image plane via homography. For example, one can select the original image in the middle as the baseline image plane and then project other original images onto it via homography to form the stitched image.

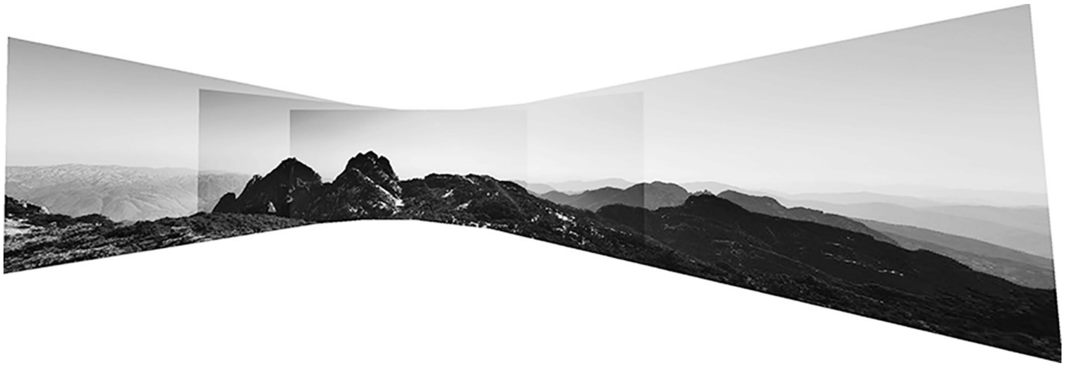


Fig. 7.13 SIFT features based image stitching (panorama)

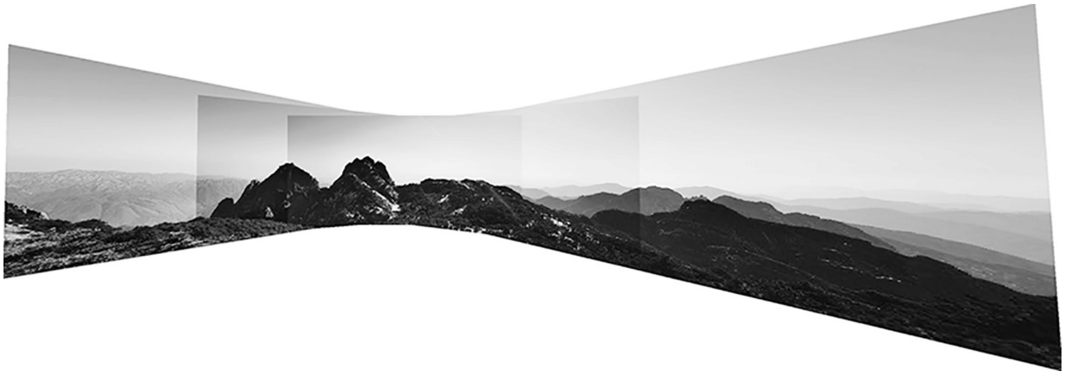


Fig. 7.14 ORB features based image stitching (panorama)

Readers would observe that the further an original image is away from the baseline image, the more stretched its projection is. Such intuitive deformation is right due to the “excessive” need to fit a panorama view onto an image plane. In fact, the original images captured from the same location but towards different view directions do form a panorama view, yet the panorama view that they form is a round view by nature. When the round view is forced to be projected onto a common image plane, the view part towards the left and right ends will inevitably be stretched. The phenomenon is illustrated in Fig. 7.15. It is like when the globe is projected onto a world map, countries towards the left and right ends are stretched and appear to be larger than what they are actually on the globe.

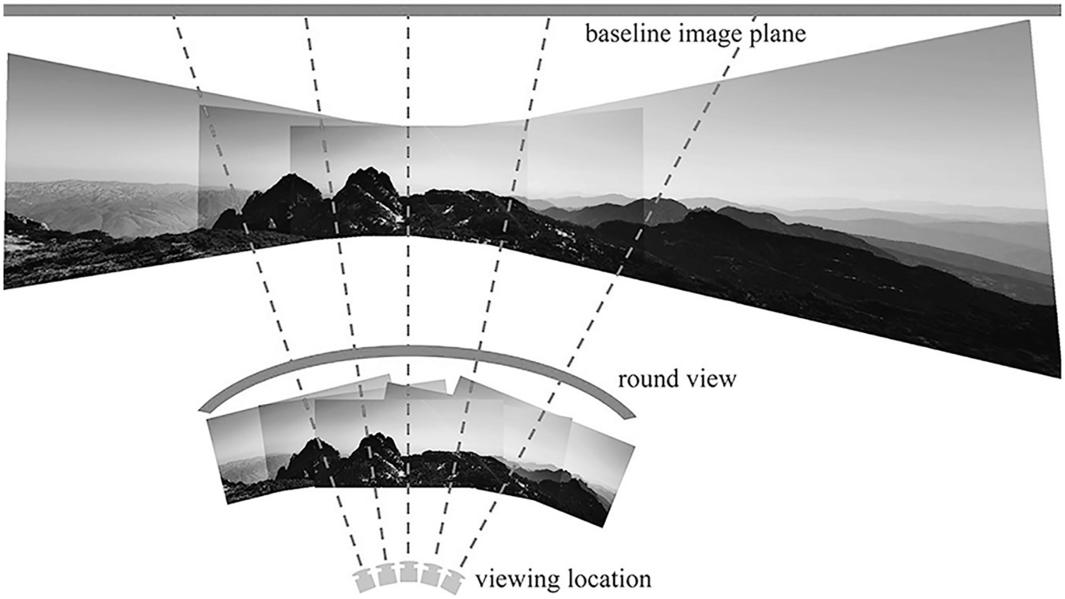


Fig. 7.15 Round view projected onto the baseline image plane

Different choices of the features such as the SIFT features and the ORB features used for image stitching will cause nuances in the stitched images, yet such nuances are not so intuitive. Readers may compare the stitched images demonstrated in Figs. 7.13 and 7.14, which seem to have no intuitive difference.

7.5 Special Tag Based Visual Localization

Special tags that facilitate image processing and pattern recognition are useful in practical applications. For example, the AprilTags [9, 10] proposed by the April Robotics Laboratory at the University of Michigan in recent years are a representative kind of special tags that can benefit vision based intelligent systems.

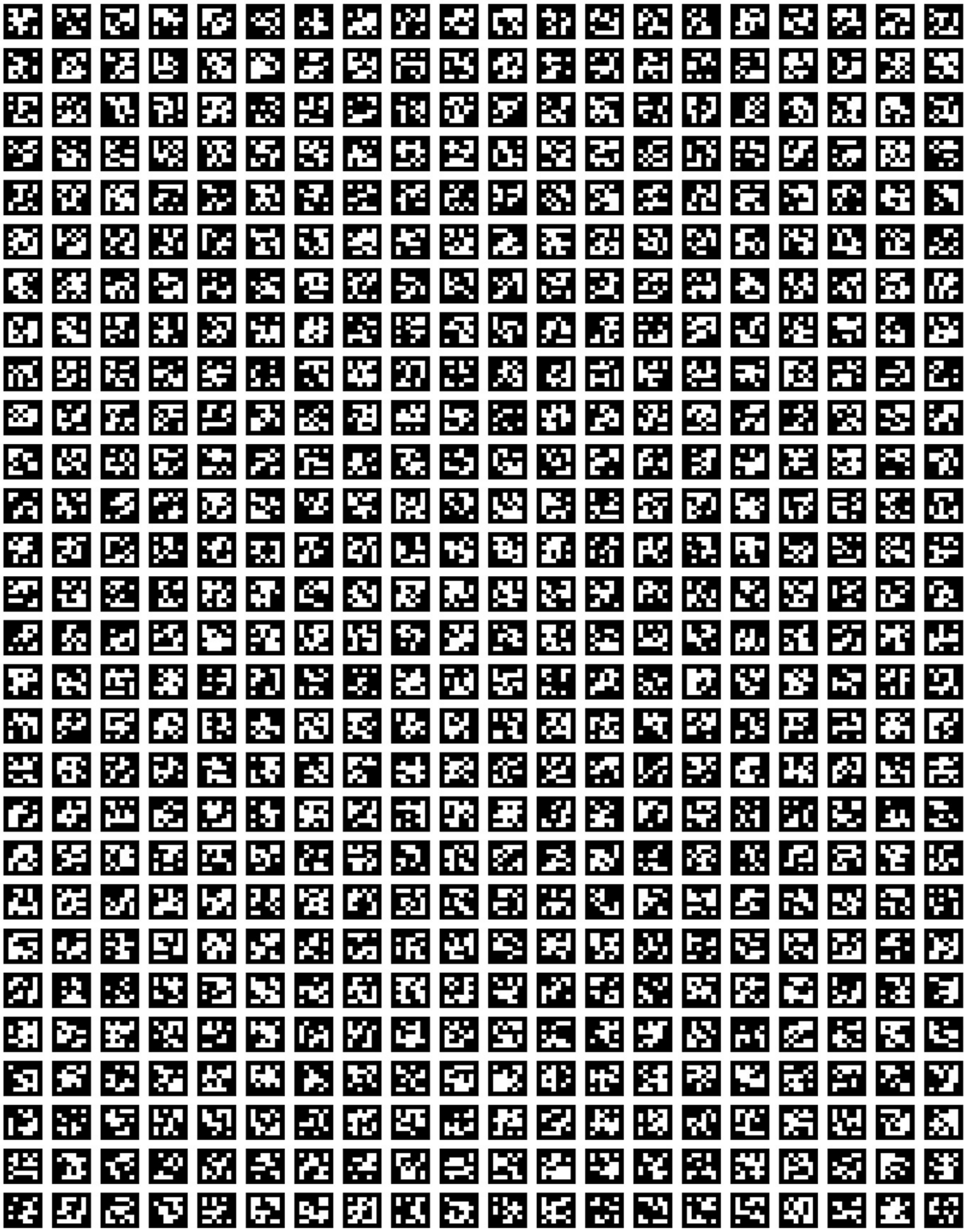


Fig. 7.16 Special tags

An AprilTag is a square-shape tag. It consists of a white outer boundary, a black inner boundary, and an internal matrix of black-white cells. For the AprilTag , itsinternal matrix of black-white cells can encode its identity. AprilTags just of a moderate internal matrix size are already enough to encode a large amount of distinct identities. For example, consider the kind of AprilTags of a 6-by-6 internal matrix. Theoretically speaking, all such AprilTags allow encoding of $2^{6\times 6} = 68\,719\,476\,736$ distinct identities. In practice, it is unnecessary to use all these AprilTags but only a small portion of them (say hundreds of them, yet already sufficient for many practical applications). Using only a small portion of all possible AprilTags has another merit, namely that the AprilTags actually adopted can be designed to have large mutual *Hamming distances* [8] among them. The larger the mutual Hamming distances among AprilTags are, the less likely they will be confused and hence the more robust the AprilTag recognition will be. A set of AprilTags are illustrated in Fig. 7.16.

AprilTags can be attached to mobile robots to distinguish their identities, which can largely facilitate data association in practical applications that involve cooperative robots. For example, a mobile robot that carries distinct AprilTags on its different sides is illustrated in Fig. 7.17a. Multiple mobile robots that carry distinct AprilTags are illustrated in Fig. 7.19a.

For detection of the AprilTag , perform the following steps:

- *Adaptive thresholding*: Perform adaptive thresholding of the original image, for which the method and the Matlab script **AdaptiveThresholding.m** are presented in Sect. 2.2.3 in Chap. 2. Then remove small foreground sets, Matlab code for which is given as follows.

RemoveSmallForeground.m

```
%% Remove small foreground sets
function imgB = RemoveSmallForeground(img, tn)
    if (nargin<2) tn = 100; end
    img([1,end],:) = 255; img(:,[1,end]) = 255;
    f = find(img==0); % Extract foreground points
    [H,W] = size(img); n = length(f); imgB = img;
    qDFS = zeros(1,n); iOff = [-H,-1,+1,+H];
    for k=1:n
        if (img(f(k))==255) continue; end
        img(f(k)) = 255; k2 = 1; qDFS(k2) = f(k); iC = [];
        while(k2>0) % DFS queue is not empty
            i = qDFS(k2); k2 = k2-1; iC = [iC, i];
            for i2=i+iOff
                if(0==img(i2))
                    k2 = k2+1; qDFS(k2) = i2; img(i2)=255;
                end
            end
        end
        if (length(iC)<tn) imgB(iC) = 255; end
    end
end
```

The binary image obtained via adaptive thresholding is demonstrated in Fig. 7.18a.

- *Edge detection and connection*: Perform edge detection on the binary image and then connect edge points into clusters, the effect of which is demonstrated in Fig. 7.18b.

- *Simple cycle extraction*: Check all edge clusters and keep those that form simple cycles while discarding the others that do not form so. The method is as follows: Consider a generic edge cluster, start from an edge point on it, take an arbitrary non-edge neighbour of the edge point, and move along the edge cluster in the direction such that the non-edge neighbour is always on the left-hand side of the moving direction. It is like moving along the left side of the road, if the edge cluster is regarded as a network of roads. To ensure moving in the desired direction, the key point is to correctly find the next edge point to traverse from current edge point, which is realized by rotating the non-edge neighbour around current edge point until an edge point neighbour is encountered. This edge point neighbour is right the next edge point to traverse. Besides, the non-edge neighbour index is updated to the last non-edge one just before encountering the edge point neighbour. The method for realizing such moving is demonstrated in the function **MoveAlongLeft**.

  Then the basic idea to determine whether an edge cluster is a simple cycle is to verify whether the trip in the move-along-left way can traverse the edge points of the cluster exactly once (i.e. with no intermediate intersection) and finally return to the starting edge point. If so, the edge cluster is extracted as a simple cycle and is kept as a quadrilateral candidate; otherwise, it is discarded. Matlab code for simple cycle extraction is given as follows.

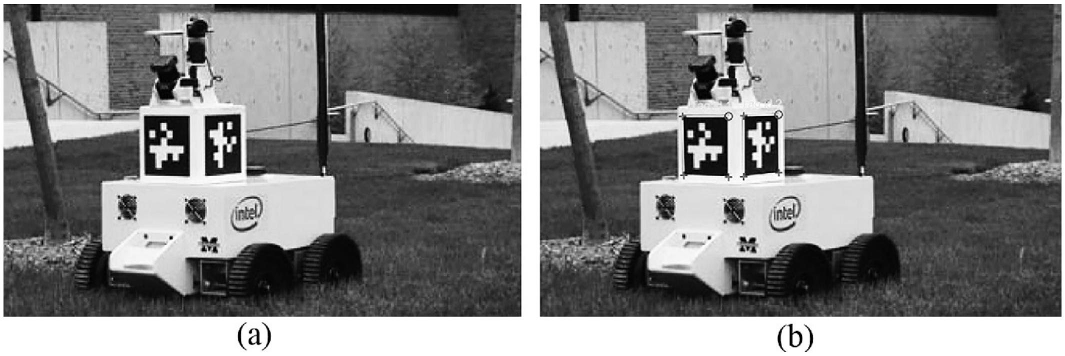
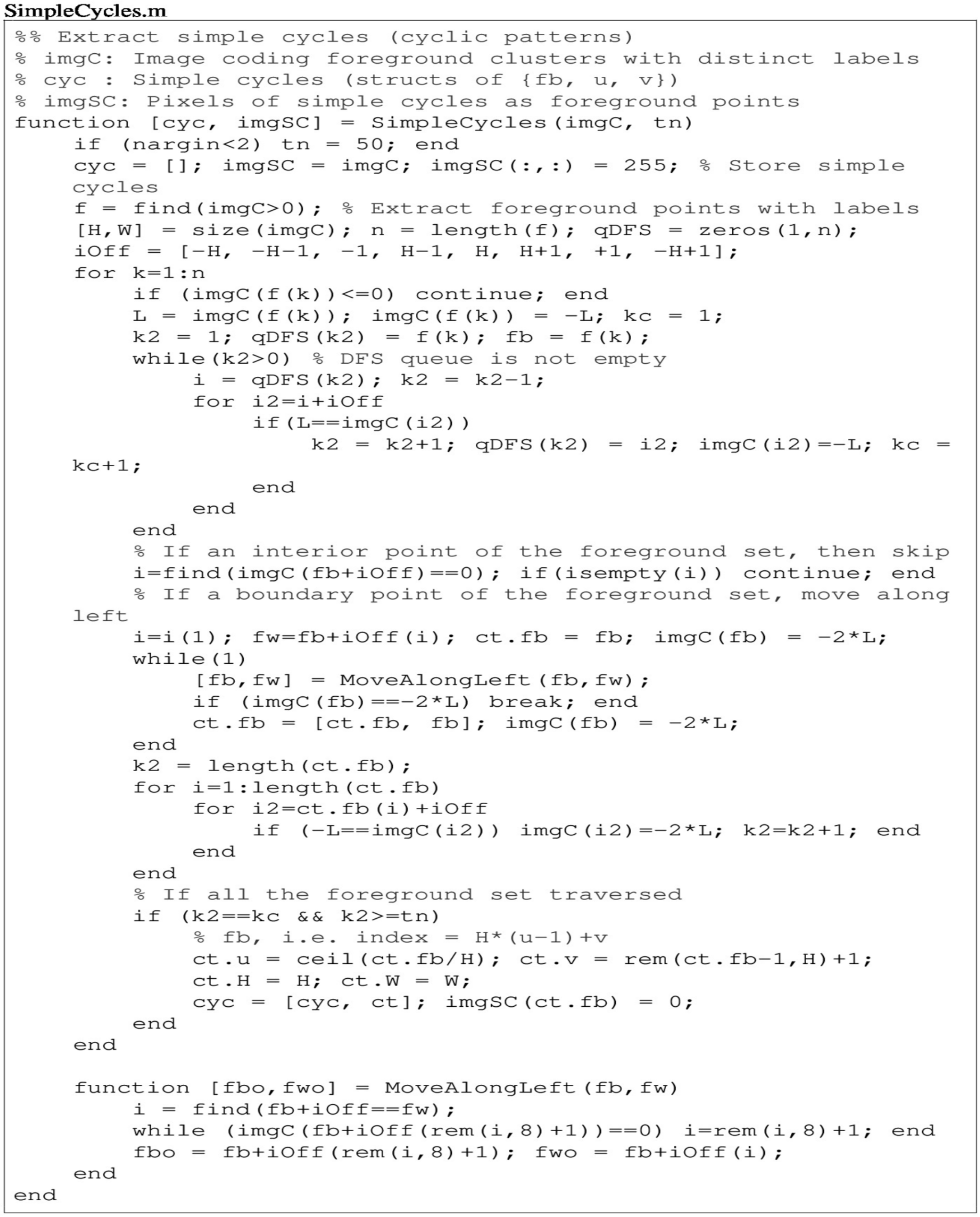

(a) (b)

Fig. 7.17 One robot carrying AprilTags **a** original image [9]; **b** AprilTag detection

SimpleCycles.m



```
%% Extract simple cycles (cyclic patterns)
% imgC: Image coding foreground clusters with distinct labels
% cyc : Simple cycles (structs of {fb, u, v})
% imgSC: Pixels of simple cycles as foreground points
function [cyc, imgSC] = SimpleCycles(imgC, tn)
    if (nargin<2) tn = 50; end
    cyc = []; imgSC = imgC; imgSC(:,:) = 255; % Store simple
    cycles
    f = find(imgC>0); % Extract foreground points with labels
    [H,W] = size(imgC); n = length(f); qDFS = zeros(1,n);
    iOff = [-H, -H-1, -1, H-1, H, H+1, +1, -H+1];
    for k=1:n
        if (imgC(f(k))<=0) continue; end
        L = imgC(f(k)); imgC(f(k)) = -L; kc = 1;
        k2 = 1; qDFS(k2) = f(k); fb = f(k);
        while(k2>0) % DFS queue is not empty
            i = qDFS(k2); k2 = k2-1;
            for i2=i+iOff
                if(L==imgC(i2))
                    k2 = k2+1; qDFS(k2) = i2; imgC(i2)=-L; kc =
    kc+1;
                end
            end
        end
        % If an interior point of the foreground set, then skip
        i=find(imgC(fb+iOff)==0); if(isempty(i)) continue; end
        % If a boundary point of the foreground set, move along
    left
        i=i(1); fw=fb+iOff(i); ct.fb = fb; imgC(fb) = -2*L;
        while(1)
            [fb,fw] = MoveAlongLeft(fb,fw);
            if (imgC(fb)==-2*L) break; end
            ct.fb = [ct.fb, fb]; imgC(fb) = -2*L;
        end
        k2 = length(ct.fb);
        for i=1:length(ct.fb)
            for i2=ct.fb(i)+iOff
                if (-L==imgC(i2)) imgC(i2)=-2*L; k2=k2+1; end
            end
        end
        % If all the foreground set traversed
        if (k2==kc && k2>=tn)
            % fb, i.e. index = H*(u-1)+v
            ct.u = ceil(ct.fb/H); ct.v = rem(ct.fb-1,H)+1;
            ct.H = H; ct.W = W;
            cyc = [cyc, ct]; imgSC(ct.fb) = 0;
        end
    end

    function [fbo,fwo] = MoveAlongLeft(fb,fw)
        i = find(fb+iOff==fw);
        while (imgC(fb+iOff(rem(i,8)+1))==0) i=rem(i,8)+1; end
        fbo = fb+iOff(rem(i,8)+1); fwo = fb+iOff(i);
    end
end
```

For sake of putting above idea into practice effectively, it is worth clarifying a number of cases. First, the starting edge point has no non-edge neighbour. This case takes place usually when the edge cluster has a blob part generated by fractal-like foreground boundaries, and implies that the edge cluster is absolutely not a simple cycle and can be discarded without initiating the trip in the move-along-left way. Second, during the trip, an edge point which has been traversed is traversed again but it is not the starting edge point. This case takes place usually when the edge cluster has an external protruding non-cyclic edge branch, and implies that the edge cluster is not a simple cycle. Third, when the starting edge point is traversed again, on the edge cluster there are still edge points neither traversed nor in the neighbourhood of traversed edge points. This case takes place usually when the edge cluster has an internal protruding non-cyclic edge branch or an internal sub-cycle, and implies that the edge cluster is not a simple cycle either. Fourth, when the starting edge point is traversed again, all the edge points of the edge cluster are either traversed or in the neighbourhood of traversed edge points. This case happens only when the edge cluster is indeed a simple cycle. Fifth, if a simple cycle is too small, then it is still discarded—It is worth noting that in the demonstration code **SimpleCycles.m**, consideration of the second case, the third case, and the fourth case is unified into a simple judgement: When an edge point which has been traversed is traversed again, is there any edge point of the edge cluster neither traversed nor in the neighbourhood of traversed edge points? If not so, the edge cluster is treated as a simple cycle[2]—Extracted simple cycles are demonstrated in Fig. 7.18c.

- *Quadrilateral verification*: Take each simple cycle as a quadrilateral candidate and verify if it is actually a quadrilateral. The method is to check whether the simple cycle has four corners exactly. Given a generic edge point $\boldsymbol{P}$ of the simple cycle, take two other edge points $\boldsymbol{P}_L$ and $\boldsymbol{P}_R$ located on the two sides of the point $\boldsymbol{P}$ respectively and with certain distance away from the point $\boldsymbol{P}$. Compute the inner product of the vectors $\boldsymbol{P}_R - \boldsymbol{P}$ and $\boldsymbol{P} - \boldsymbol{P}_L$. The smaller the inner product is, the more like a corner the point $\boldsymbol{P}$ is. Compute inner products corresponding to edge points of the simple cycle, apply non-minimum suppression to the inner products. If there are exactly four inner product valleys below certain threshold (say 0.5), then the simple cycle is verified as a quadrilateral and kept; otherwise, discarded.

Quadrilaterals.m

```
%% Verify which simple cycles are quadrilaterals
function [qd, imgQd] = Quadrilaterals(cyc)
    qd = []; sw = 5; ipT = 0.5;
    imgQd = zeros(cyc(1).H,cyc(1).W); imgQd(:,:) = 255;
    for k=1:length(cyc)
        u = cyc(k).u; v = cyc(k).v; n = length(u);
        ip = zeros(1,n); % List of inner products
        for k2=1:n
            k2L = rem(k2+n-sw-1,n)+1; k2R = rem(k2+sw-1,n)+1;
            duL = u(k2)-u(k2L); duR = u(k2R)-u(k2);
            dvL = v(k2)-v(k2L); dvR = v(k2R)-v(k2);
            % Inner product: the smaller, the more like a corner
            ip(k2) = (duL*duR+dvL*dvR)/(sw^2);
        end
        % Non-minimum suppression
        iporg = ip;
        for k2=1:n
            k2L = rem(k2+n-2,n)+1; k2R = rem(k2,n)+1;
            % Prefer left when equal
            if (iporg(k2)>=iporg(k2L) || iporg(k2)>iporg(k2R))
                ip(k2) = 1;
            end
        end
        i = find(ip<ipT); % Indices of potential corners
        if (length(i)~= 4) continue; end
        % Refine corner localization
        qde(1).i=i(1):i(2); qde(2).i=i(2):i(3);
        qde(3).i=i(3):i(4); qde(4).i=[i(4):n,1:i(1)];
        abc = zeros(3,4); uv = zeros(2,4);
        for k2=1:4
            abc(:,k2) = LineFitting2D_II(u(qde(k2).i),v(qde(k2).
    i));
        end
        for k2=1:4
            uv(:,k2)=-abc(1:2,[k2,rem(k2,4)+1])’\abc(3,[k2,rem(
    k2,4)+1])’;
        end
        qd = [qd; reshape(uv,1,[])]; imgQd(cyc(k).fb) = 0;
    end
end
```

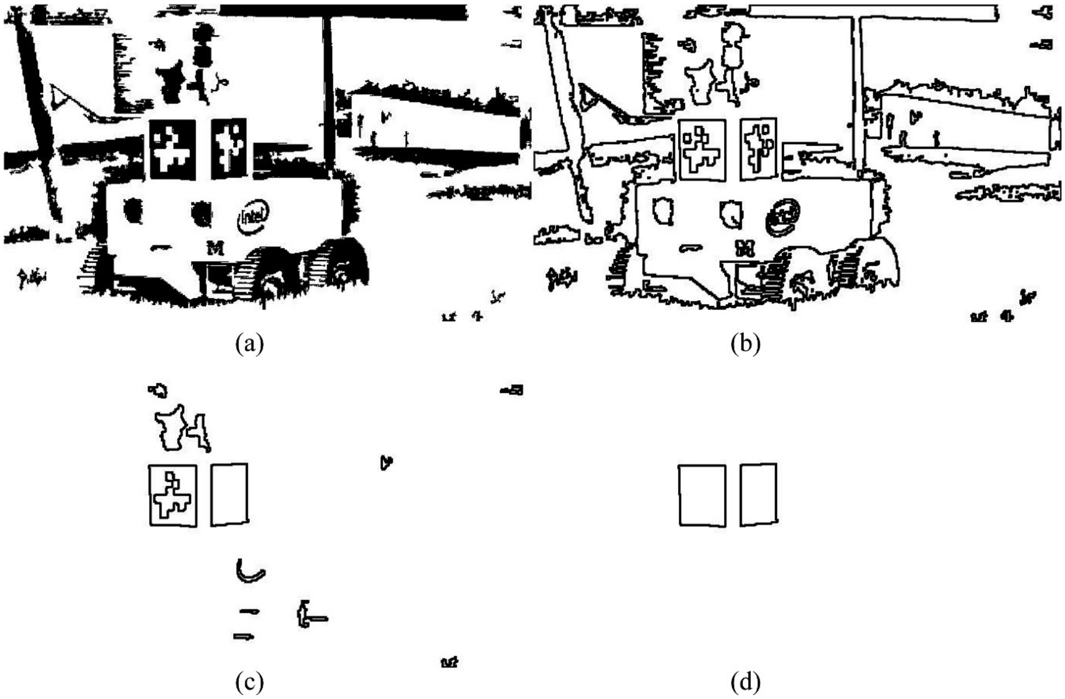

(a) (b) (c) (d)

Fig. 7.18 Intermediate results of AprilTag detection **a** binary image after adaptive thresholding; **b** connected edges; **c** simple cycles; **d** quadrilaterals

Simple cycles that are verified as quadrilaterals are demonstrated in Fig. 7.18d. To refine localization of the four corners of the quadrilateral, one can apply line fitting to each side of the quadrilateral and take intersections of the four fitted lines as the refined results. The sub-function **LineFitting2D_II** for line fitting has already been given in Sect. 6.2.1 in Chap. 6.

For decoding of the AprilTag , perform the following steps:

- Suppose the AprilTag consists of $N \times N$ cells (a conventionally adopted $N = 8$). The boundary cells are all black cells, whereas the internal $(N-2) \times (N-2)$ matrix of black-white cells are the AprilTag part actually used to encode its identity. For the AprilTag , revealing its internal $(N-2) \times (N-2)$ matrix of black-white cells is right what AprilTag decoding means. For the extracted four corners of the AprilTag quadrilateral, suppose their image coordinates are

$$(u_1, v_1), \quad (u_2, v_2), \quad (u_3, v_3), \quad (u_4, v_4)$$

and their world coordinates on the AprilTag plane are

$$\begin{aligned}(x_{\mathrm{w},1}, y_{\mathrm{w},1}) &= (N, 0), \quad (x_{\mathrm{w},2}, y_{\mathrm{w},2}) = (N, N),\\ (x_{\mathrm{w},3}, y_{\mathrm{w},3}) &= (0, N), \quad (x_{\mathrm{w},4}, y_{\mathrm{w},4}) = (0, 0),\end{aligned}$$

where the cell width is simply treated as the world coordinate unit. For the homography between the AprilTag plane and the image coordinates system

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{\mathrm{w}} \\ y_{\mathrm{w}} \\ 1 \end{bmatrix},$$

calibrate the homography matrix $\boldsymbol{H}$ by solving the homogeneous linear equation (4.23) or equivalently (4.24)

$$\begin{bmatrix} x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & 0 & 0 & 0 & -u_1 x_{\mathrm{w},1} & -u_1 y_{\mathrm{w},1} & -u_1 \\ 0 & 0 & 0 & x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & -v_1 x_{\mathrm{w},1} & -v_1 y_{\mathrm{w},1} & -v_1 \\ x_{\mathrm{w},2} & y_{\mathrm{w},2} & 1 & 0 & 0 & 0 & -u_2 x_{\mathrm{w},2} & -u_2 y_{\mathrm{w},2} & -u_2 \\ 0 & 0 & 0 & x_{\mathrm{w},2} & y_{\mathrm{w},2} & 1 & -v_2 x_{\mathrm{w},2} & -v_2 y_{\mathrm{w},2} & -v_2 \\ x_{\mathrm{w},3} & y_{\mathrm{w},3} & 1 & 0 & 0 & 0 & -u_3 x_{\mathrm{w},3} & -u_3 y_{\mathrm{w},3} & -u_3 \\ 0 & 0 & 0 & x_{\mathrm{w},3} & y_{\mathrm{w},3} & 1 & -v_3 x_{\mathrm{w},3} & -v_3 y_{\mathrm{w},3} & -v_3 \\ x_{\mathrm{w},4} & y_{\mathrm{w},4} & 1 & 0 & 0 & 0 & -u_4 x_{\mathrm{w},4} & -u_4 y_{\mathrm{w},4} & -u_4 \\ 0 & 0 & 0 & x_{\mathrm{w},4} & y_{\mathrm{w},4} & 1 & -v_4 x_{\mathrm{w},4} & -v_4 y_{\mathrm{w},4} & -v_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \mathbf{0}.$$

- Use the calibrated homography matrix to project the internal matrix of black-white cells onto the image plane to obtain their corresponding image parts and compute the average pixel values of these image parts respectively. More specifically, given a generic internal cell with four corners

  $$(c, r), \quad (c+1, r), \quad (c+1, r+1), \quad (c, r+1),$$

  where $c, r \in \{1, \ldots, N-2\}$. Compute their corresponding image coordinates

  $$(u_{c,r}, v_{c,r}), \quad (u_{c+1,r}, v_{c+1,r}), \quad (u_{c+1,r+1}, v_{c+1,r+1}), \quad (u_{c,r+1}, v_{c,r+1})$$

  via homography respectively. Sort $\{u_{c,r}, u_{c+1,r}, u_{c+1,r+1}, u_{c,r+1}\}$ in ascending order as $u^{[1]}_{[c,c+1]} \le u^{[2]}_{[c,c+1]} \le u^{[3]}_{[c,c+1]} \le u^{[4]}_{[c,c+1]}$ and select the intermediate two ones $\{u^{[2]}_{[c,c+1]}, u^{[3]}_{[c,c+1]}\}$. Sort $\{v_{c,r}, v_{c+1,r}, v_{c+1,r+1}, v_{c,r+1}\}$ in ascending order as $v^{[1]}_{[r,r+1]} \le v^{[2]}_{[r,r+1]} \le v^{[3]}_{[r,r+1]} \le v^{[4]}_{[r,r+1]}$ and select the intermediate two ones $\{v^{[2]}_{[r,r+1]}, v^{[3]}_{[r,r+1]}\}$. Take the image patch enclosed by the four image points

$$(u^{[2]}_{[c,c+1]}, v^{[2]}_{[r,r+1]}),\ (u^{[3]}_{[c,c+1]}, v^{[2]}_{[r,r+1]}),\ (u^{[3]}_{[c,c+1]}, v^{[3]}_{[r,r+1]}),\ (u^{[2]}_{[c,c+1]}, v^{[3]}_{[r,r+1]})$$

and assign the average pixel value of the image patch to the generic internal cell.

- After previous steps, the internal $(N-2) \times (N-2)$ matrix is assigned $(N-2) \times (N-2)$ pixel values corresponding to its cells respectively. Compute their mean pixel value and use this mean value to threshold the internal matrix to obtain the final AprilTag decoding result. Matlab code for AprilTag decoding is given as follows.

DecodeAprilTags.m

```
%% Decode AprilTags into binary matrices
function tgC = DecodeAprilTags(qduv,img,tgN)
    Hm = CalibrateHomography(qduv(1:2:end), qduv(2:2:end), ...
        [tgN,tgN,0,0], [0,tgN,tgN,0]);
    xw = reshape(repmat([0:tgN],tgN+1,1),1,[]);
    yw = reshape(repmat([0:tgN]’,1,tgN+1),1,[]);
    [u, v] = HomographyXY2UV(Hm, xw, yw); tgC = zeros(tgN,tgN);
    u = reshape(round(u),tgN+1,[]); v = reshape(round(v),tgN
    +1,[]);
    for r=1:tgN
        for c=1:tgN
            utmp = sort(reshape(u(r:r+1,c:c+1),1,[]));
            vtmp = sort(reshape(v(r:r+1,c:c+1),1,[]));
            tgC(r,c) = mean(mean(img(vtmp(2):vtmp(3),utmp(2):
    utmp(3))));
        end
    end
    tgC = tgC(2:end-1,2:end-1); % Cut black boundary
    iB = find(tgC<mean(mean(tgC))); tgC(:,:) = 1; tgC(iB) = 0;
end
```

It is worth noting that in practice, computation of the mean pixel values of the image patches can take advantage of the integral image already established for adaptive thresholding, though in above Matlab script **DecodeAprilTags.m** the built-in function *mean* is directly used for demo code conciseness.

For the two AprilTags attached to the mobile robot illustrated in Fig. 7.17, their decoding results are respectively

$$\text{Tag 1code}: \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Tag 2code}: \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

where 1 denotes white and 0 denotes black. As shown above, the decoding results are consistent with the configurations of black-white cells of the two AprilTags .

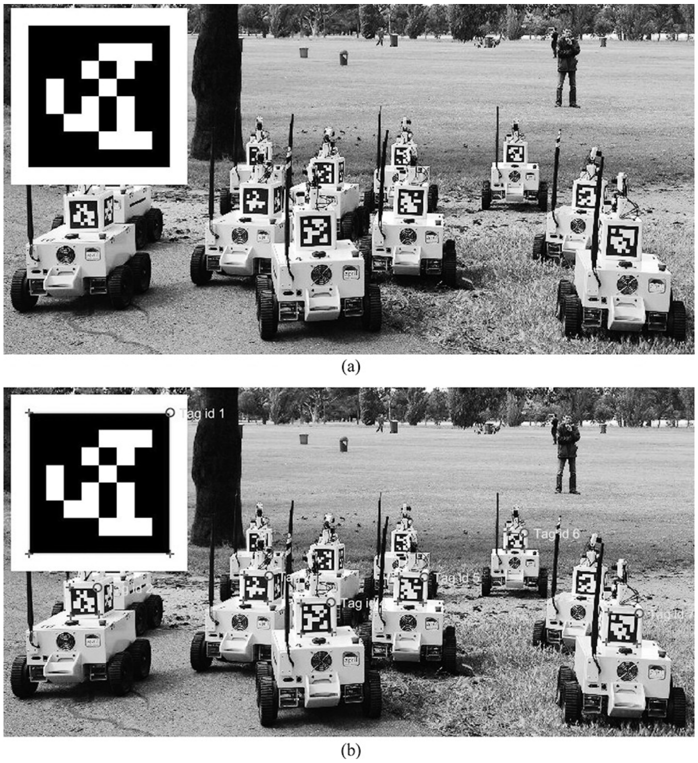
Matlab code for complete detection and decoding of AprilTags is given as follows. For the image illustrated in Fig. 7.17a, the AprilTag detection results are visualized in Fig. 7.17b.

DetectAprilTag.m

```
%% Detect AprilTags
% qd  : [u11,v11,u12,v12,u13,v13,u14,v14; (first quadrilateral)
%     : u21,v21,u22,v22,u23,v23,u24,v24; (second quadrilateral)
    ...]
% tg  : Tensor of tag codes (6$\,\times\,$6 x qd_size1)
% imgB : Images after adaptive thresholding
% imgC : Connected object edges
% imgSC: Pixels of simple cycles as foreground points
% imgQd: Pixels of quadrilaterals
function [qd, tg, imgB, imgC, imgSC, imgQd] = DetectAprilTag(img
    )
    [H,W] = size(img);
    imgB = AdaptiveThresholding(img);
    imgB = RemoveSmallForeground(imgB);
    imgE = zeros(H,W); imgE(:,:) = 255; Im = zeros(H,W);
    Im(:,2:end) = Im(:,2:end)+abs(imgB(:,2:end)-imgB(:,1:end-1))
    ;
    Im(2:end,:) = Im(2:end,:)+abs(imgB(2:end,:)-imgB(1:end-1,:))
    ;
    i = find(Im>=255); imgE(i) = 0;
    imgC = DirectConnection(imgE);
    [cyc, imgSC] = SimpleCycles(imgC);
    [qd, imgQd] = Quadrilaterals(cyc);
    tg = zeros(6,6,size(qd,1));
    for k=1:size(qd,1) % Decode AprilTags
        tg(:,:,k) = DecodeAprilTags(qd(k,:),img,8);
    end
end
```

Another image example is illustrated in Fig. 7.19a, its AprilTag detection results are visualized in Fig. 7.19b. As one can see, all AprilTags that are not occluded are detected correctly.

AprilTags are especially useful to vision based mobile robotics. For example, they can serve for visual localization based warehouse robot navigation [11].[3] More specifically, one can install AprilTags in the environment where the mobile robot operates and establish an AprilTag map dedicated to robot visual localization. To enhance accuracy of AprilTag mapping , one can take advantage of a LiDAR SLAM (simultaneous localization and mapping) system to accurately estimate poses of the mapping robot, namely the unique mobile robot equipped with the LiDAR SLAM system. Then the AprilTag map namely poses of the AprilTags in the environment can be accurately estimated based on estimated poses of the mapping robot, as illustrated in Fig. 7.20. This AprilTag map established by the mapping robot can be shared by a large amount of visual localization based mobile robots. The method of visual localization based on the accurate AprilTag map is illustrated in Fig. 7.21.


Tag id 1
Tag id 6

(a)

(b)

Fig. 7.19 Mobile robots carrying AprilTags **a** original image [9]; **b** AprilTag detection

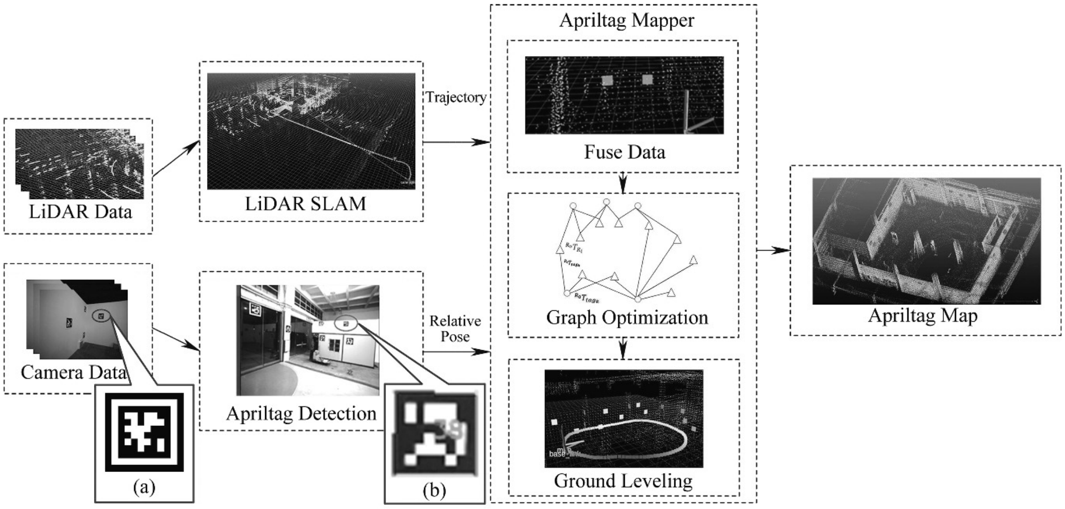
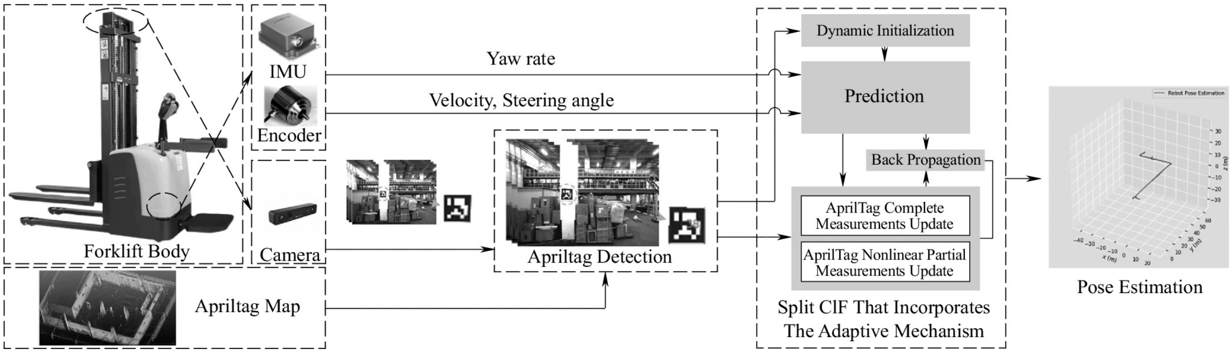

LiDAR Data
LiDAR SLAM
Trajectory
Camera Data
Apriltag Detection
Relative Pose
(a)
(b)
Apriltag Mapper
Fuse Data
Graph Optimization
Ground Leveling
Apriltag Map

Fig. 7.20 AprilTag mapping [11]

No matter for AprilTag mapping and for AprilTag map based visual localization, a key point consists in estimation of mobile robot-AprilTag relative poses. Given a mobile robot and an AprilTag that the mobile robot can detect, the homography matrix $\boldsymbol{H}$ between the AprilTag plane and the image plane of the camera installed on the mobile robot can be computed by solving the homogeneous linear equation (4.23) or equivalently (4.24).



Forklift Body
IMU
Encoder
Camera
Apriltag Map
Yaw rate
Velocity, Steering angle
Apriltag Detection
Dynamic Initialization
Prediction
Back Propagation
AprilTag Complete Measurements Update
AprilTag Nonlinear Partial Measurements Update
Split CIF That Incorporates The Adaptive Mechanism
Robot Pose Estimation
Pose Estimation

Fig. 7.21 AprilTag map based visual localization [11]

Suppose the camera intrinsic matrix $\boldsymbol{A}$ is calibrated. Further suppose the rotation matrix $\boldsymbol{R}_{\mathrm{cr}}$ and the translation vector $\boldsymbol{T}_{\mathrm{cr}}$ from the camera coordinates system to the robot coordinates system are also calibrated. The relative pose between the AprilTag and the camera coordinates system, namely the rotation matrix $\boldsymbol{R}_{\mathrm{wc}}$ and the translation vector $\boldsymbol{T}_{\mathrm{wc}}$ from the AprilTag coordinates system to the camera coordinates system, can be computed via (5.2) and (5.40):

$$\begin{aligned}
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1: &= \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}\|_2},\\
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2: &= \frac{\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}}{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}\|_2},\\
\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_3: &= (\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1)\times(\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2),\\
\boldsymbol{w}_{\mathrm{wc}} &= \boldsymbol{R}^{-1}(\boldsymbol{R}_{\mathrm{wc}}) = \boldsymbol{R}^{-1}([\boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_1 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_2 \quad \boldsymbol{R}_{\mathrm{wc}}\boldsymbol{e}_3]),\\
\boldsymbol{R}_{\mathrm{wc}}: &= \boldsymbol{R}(\boldsymbol{w}_{\mathrm{wc}}) \qquad \text{(update the rotation matrix)},\\
\lambda &= \sqrt{\|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,1}\|_2 \cdot \|\boldsymbol{A}^{-1}\boldsymbol{H}_{:,2}\|_2},\\
\boldsymbol{T}_{\mathrm{wc}} &= \tfrac{1}{\lambda}\boldsymbol{A}^{-1}\boldsymbol{H}_{:,3},
\end{aligned}$$

where $\boldsymbol{R}(\cdot)$ denotes the rotation function that maps a rotation vector to its associated rotation matrix and $\boldsymbol{R}^{-1}(\cdot)$ denotes the inverse rotation function that maps a (quasi)rotation matrix to its associated rotation vector via the method presented in Sect. exrefsec:rodriguesspsformula in Chap. 5. Then the relative pose between the AprilTag and the mobile robot can be computed as

$$\begin{bmatrix} \boldsymbol{R}_{\mathrm{wr}} & \boldsymbol{T}_{\mathrm{wr}} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{\mathrm{cr}} & \boldsymbol{T}_{\mathrm{cr}} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{\mathrm{wc}} & \boldsymbol{T}_{\mathrm{wc}} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \tag{7.14}$$
$$\Leftrightarrow \quad \boldsymbol{R}_{\mathrm{wr}} = \boldsymbol{R}_{\mathrm{cr}}\boldsymbol{R}_{\mathrm{wc}}, \quad \boldsymbol{T}_{\mathrm{wr}} = \boldsymbol{R}_{\mathrm{cr}}\boldsymbol{T}_{\mathrm{wc}} + \boldsymbol{T}_{\mathrm{cr}}.$$

Singularity of homography matrix computation

The homogeneous linear equation (4.23) or equivalently (4.24)

$$\begin{bmatrix} x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & 0 & 0 & 0 & -u_1x_{\mathrm{w},1} & -u_1y_{\mathrm{w},1} & -u_1 \\ 0 & 0 & 0 & x_{\mathrm{w},1} & y_{\mathrm{w},1} & 1 & -v_1x_{\mathrm{w},1} & -v_1y_{\mathrm{w},1} & -v_1 \\ x_{\mathrm{w},2} & y_{\mathrm{w},2} & 1 & 0 & 0 & 0 & -u_2x_{\mathrm{w},2} & -u_2y_{\mathrm{w},2} & -u_2 \\ 0 & 0 & 0 & x_{\mathrm{w},2} & y_{\mathrm{w},2} & 1 & -v_2x_{\mathrm{w},2} & -v_2y_{\mathrm{w},2} & -v_2 \\ x_{\mathrm{w},3} & y_{\mathrm{w},3} & 1 & 0 & 0 & 0 & -u_3x_{\mathrm{w},3} & -u_3y_{\mathrm{w},3} & -u_3 \\ 0 & 0 & 0 & x_{\mathrm{w},3} & y_{\mathrm{w},3} & 1 & -v_3x_{\mathrm{w},3} & -v_3y_{\mathrm{w},3} & -v_3 \\ x_{\mathrm{w},4} & y_{\mathrm{w},4} & 1 & 0 & 0 & 0 & -u_4x_{\mathrm{w},4} & -u_4y_{\mathrm{w},4} & -u_4 \\ 0 & 0 & 0 & x_{\mathrm{w},4} & y_{\mathrm{w},4} & 1 & -v_4x_{\mathrm{w},4} & -v_4y_{\mathrm{w},4} & -v_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \boldsymbol{0}$$

for solving the homography matrix $\boldsymbol{H}$ may become ill-conditioned or even singular due to AprilTag detection errors. This may happen normally when the AprilTag is far away from the mobile robot. As a consequence, the obtained homography matrix $\boldsymbol{H}$ may be errant and hence so is the estimated mobile robot-AprilTag relative pose.

On the other hand, even when such undesirable ill-conditioned case takes place, despite uselessness of the wrong estimate of the mobile robot-AprilTag relative pose $\{\boldsymbol{R}_{\mathrm{wr}}, \boldsymbol{T}_{\mathrm{wr}}\}$, the revealed distance between the AprilTag and the camera, i.e. $\|\boldsymbol{R}_{\mathrm{wc}}\|$ is still useful to visual localization of the mobile robot [11].

7.6 Visual SLAM

Geometry fundamentals of computer vision are indispensable to **visual SLAM** namely **visual simultaneous localization and mapping** [12–14]. The objective of visual SLAM is to reveal three-dimensional structures of the environment from multiple images (views) of the environment.

If one knows how to reveal three-dimensional structures of the environment from two images, then one can naturally extend the method to reveal three-dimensional structures of the environment from multiple images. To reveal three-dimensional structures of the environment from two images, one needs to handle the core problem of how to estimate the inter-camera relative pose[4] from the pair of images.

For the purpose, resort to knowledge on the epipolar constraint presented in Sect. 4.5.2 in Chap. 4. Given two generic images of a roughly common part of the environment, consider the epipolar constraint (4.65) between the pair of images

$$\begin{bmatrix} x_{\mathrm{c}}^{[1]} & y_{\mathrm{c}}^{[1]} & z_{\mathrm{c}}^{[1]} \end{bmatrix} \begin{bmatrix} \alpha^{[1]} & 0 & 0 \\ \gamma^{[1]} & \beta^{[1]} & 0 \\ u_0^{[1]} & v_0^{[1]} & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha^{[2]} & \gamma^{[2]} & u_0^{[2]} \\ 0 & \beta^{[2]} & v_0^{[2]} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}}^{[2]} \\ y_{\mathrm{c}}^{[2]} \\ z_{\mathrm{c}}^{[2]} \end{bmatrix} = 0,$$

i.e.

$$\boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{E} \boldsymbol{P}_{\mathrm{c}}^{[2]} = \boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{A}^{[1]\mathrm{T}} \boldsymbol{F} \boldsymbol{A}^{[2]} \boldsymbol{P}_{\mathrm{c}}^{[2]} = 0.$$

In the context of visual SLAM , the pair of images are captured by the same camera, so the camera intrinsic matrices $\boldsymbol{A}^{[1]}$ and $\boldsymbol{A}^{[2]}$ actually refer to the same camera intrinsic matrix $\boldsymbol{A}$. Denote it as

$$\boldsymbol{A}^{[1]} = \boldsymbol{A}^{[2]} = \boldsymbol{A} \equiv \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

and the epipolar constraint (4.65) becomes

$$\begin{bmatrix} x_{\mathrm{c}}^{[1]} & y_{\mathrm{c}}^{[1]} & z_{\mathrm{c}}^{[1]} \end{bmatrix} \begin{bmatrix} \alpha & 0 & 0 \\ \gamma & \beta & 0 \\ u_0 & v_0 & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{c}}^{[2]} \\ y_{\mathrm{c}}^{[2]} \\ z_{\mathrm{c}}^{[2]} \end{bmatrix} = 0 \qquad (7.15)$$

or compactly as

$$\boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{E} \boldsymbol{P}_{\mathrm{c}}^{[2]} = \boldsymbol{P}_{\mathrm{c}}^{[1]\mathrm{T}} \boldsymbol{A}^{\mathrm{T}} \boldsymbol{F} \boldsymbol{A} \boldsymbol{P}_{\mathrm{c}}^{[2]} = 0.$$

Suppose the camera intrinsic matrix $\boldsymbol{A}$ is calibrated *a priori* and hence known.

Compute the essential matrix

$$\boldsymbol{E} = \boldsymbol{A}^{\mathrm{T}} \boldsymbol{F} \boldsymbol{A}$$

using the method presented in Sect. 5.4.2 in Chap. 5. Recall (4.73)

$$\boldsymbol{E} \propto \boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$$

via which the rotation matrix $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$ and the translation vector $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ that characterize the inter-camera relative pose can be recovered from the computed essential matrix $\boldsymbol{E}$.

More specifically, neglect the scaling factor involved in (4.73) and just analyse the equation

$$\boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{E} \tag{7.16}$$

in terms of the translation vector $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ and the rotation matrix $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$ or the rotation vector $\boldsymbol{w}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{R}^{-1}(\boldsymbol{R}_{\mathrm{cc}}^{[2,1]})$. Here, $\boldsymbol{R}^{-1}(\cdot)$ denotes the inverse rotation function that maps a (quasi) rotation matrix to its associated rotation vector via the method presented in Sect. 5.2.1 in Chap. 5. Once the solution of (7.16) is obtained, then add back the scaling factor to the solution of $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ according to *ad hoc* field knowledge or conventionally normalize the solution of $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ for sake of visual SLAM initialization.

Compute the translation vector $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$

Denote $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \equiv \begin{bmatrix} x_{\mathrm{cc}}^{[2,1]} & y_{\mathrm{cc}}^{[2,1]} & z_{\mathrm{cc}}^{[2,1]} \end{bmatrix}^{\mathrm{T}}$. According to (4.75), perform singular value decomposition of $\boldsymbol{E}$ as $\boldsymbol{E} = \boldsymbol{U}\mathrm{diag}(\sigma, \sigma, 0)\boldsymbol{V}^{\mathrm{T}}$ or in details as

$$\boldsymbol{E} = [\boldsymbol{U}_1 \quad \boldsymbol{U}_2 \quad \boldsymbol{U}_3] \begin{bmatrix} \sigma & & \\ & \sigma & \\ & & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^{\mathrm{T}} \\ \boldsymbol{V}_2^{\mathrm{T}} \\ \boldsymbol{V}_3^{\mathrm{T}} \end{bmatrix}.$$

Substitute above equality into (7.16) and obtain

$$\begin{aligned} \boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U}\mathrm{diag}(\sigma, \sigma, 0)\boldsymbol{V}^{\mathrm{T}} &\iff \boldsymbol{U}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} \boldsymbol{V} = \mathrm{diag}(\sigma, \sigma, 0) \\ &\iff \boldsymbol{U}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{U}\boldsymbol{U}^{\mathrm{T}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} \boldsymbol{V} = \mathrm{diag}(\sigma, \sigma, 0) \end{aligned}$$

namely

$$\boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} = \mathrm{diag}(\sigma, \sigma, 0), \tag{7.17}$$

where

$$\boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \equiv \boldsymbol{U}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{U}, \tag{7.18a}$$

$$\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \equiv \boldsymbol{U}^{\mathrm{T}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} \boldsymbol{V}. \tag{7.18b}$$

Denote $\bar{\mathbf{T}}_{\mathrm{cc}}^{[2,1]} \equiv \left[\bar{x}_{\mathrm{cc}}^{[2,1]} \quad \bar{y}_{\mathrm{cc}}^{[2,1]} \quad \bar{z}_{\mathrm{cc}}^{[2,1]}\right]^{\mathrm{T}}$. It can be verified that $\boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}}$ is a skew-symmetric matrix, whereas $\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]}$ is a rotation matrix.

From (7.17) one has

$$
\begin{aligned}
\begin{bmatrix} \sigma^2 & & \\ & \sigma^2 & \\ & & 0 \end{bmatrix} &= \boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \left(\boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]}\right)^{\mathrm{T}} = \boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}}^{\mathrm{T}} \\
&= \begin{bmatrix} \left(\bar{y}_{\mathrm{cc}}^{[2,1]}\right)^2 + \left(\bar{z}_{\mathrm{cc}}^{[2,1]}\right)^2 & -\bar{x}_{\mathrm{cc}}^{[2,1]} \bar{y}_{\mathrm{cc}}^{[2,1]} & -\bar{x}_{\mathrm{cc}}^{[2,1]} \bar{z}_{\mathrm{cc}}^{[2,1]} \\ -\bar{x}_{\mathrm{cc}}^{[2,1]} \bar{y}_{\mathrm{cc}}^{[2,1]} & \left(\bar{x}_{\mathrm{cc}}^{[2,1]}\right)^2 + \left(\bar{z}_{\mathrm{cc}}^{[2,1]}\right)^2 & -\bar{y}_{\mathrm{cc}}^{[2,1]} \bar{z}_{\mathrm{cc}}^{[2,1]} \\ -\bar{x}_{\mathrm{cc}}^{[2,1]} \bar{z}_{\mathrm{cc}}^{[2,1]} & -\bar{y}_{\mathrm{cc}}^{[2,1]} \bar{z}_{\mathrm{cc}}^{[2,1]} & \left(\bar{x}_{\mathrm{cc}}^{[2,1]}\right)^2 + \left(\bar{y}_{\mathrm{cc}}^{[2,1]}\right)^2 \end{bmatrix},
\end{aligned}
$$

which implies

$$
\bar{x}_{\mathrm{cc}}^{[2,1]} = 0, \quad \bar{y}_{\mathrm{cc}}^{[2,1]} = 0, \quad \left(\bar{z}_{\mathrm{cc}}^{[2,1]}\right)^2 = \sigma^2. \tag{7.19}
$$

So

$$
\bar{z}_{\mathrm{cc}}^{[2,1]} = \sigma \Rightarrow \bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]} = [0 \quad 0 \quad \sigma]^{\mathrm{T}} \tag{7.20}
$$

or

$$
\bar{z}_{\mathrm{cc}}^{[2,1]} = -\sigma \Rightarrow \bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]} = [0 \quad 0 \quad -\sigma]^{\mathrm{T}}. \tag{7.21}
$$

Once $\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}$ is obtained, apply the first equation of (7.18) to recover $\boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}}$ as

$$
\boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} = \boldsymbol{U} \boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{U}^{\mathrm{T}} = \boldsymbol{S}_{\boldsymbol{U}\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} \tag{7.22}
$$

and then recover the translation vector $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}$ as $\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U}\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}$.

Compute the rotation matrix $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$

Substitute the following equality

$$
\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]} = \begin{bmatrix} 0 \\ 0 \\ \bar{z}_{\mathrm{cc}}^{[2,1]} \end{bmatrix} \Longleftrightarrow \boldsymbol{S}_{\bar{\boldsymbol{T}}_{\mathrm{cc}}^{[2,1]}} = \begin{bmatrix} 0 & -\bar{z}_{\mathrm{cc}}^{[2,1]} & 0 \\ \bar{z}_{\mathrm{cc}}^{[2,1]} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
$$

into (7.17) and obtain

$$\begin{bmatrix} 0 & -\bar{z}_{\mathrm{cc}}^{[2,1]} & 0 \\ \bar{z}_{\mathrm{cc}}^{[2,1]} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \\ \boldsymbol{e}_2^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \\ \boldsymbol{e}_3^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \end{bmatrix} = \begin{bmatrix} \sigma & & \\ & \sigma & \\ & & 0 \end{bmatrix}$$

which implies

$$\begin{aligned} \boldsymbol{e}_1^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} &= [0 \sigma / \bar{z}_{\mathrm{cc}}^{[2,1]} \quad 0], \\ \boldsymbol{e}_2^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} &= [-\sigma / \bar{z}_{\mathrm{cc}}^{[2,1]} \quad 0 \quad 0], \\ \boldsymbol{e}_3^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} = (\boldsymbol{e}_1^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]}) \times (\boldsymbol{e}_2^{\mathrm{T}} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]}) &= [0 \quad 0 \quad \sigma^2 / (\bar{z}_{\mathrm{cc}}^{[2,1]})^2] = [0 \quad 0 \quad 1] \end{aligned}$$

namely

$$\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} = \begin{bmatrix} 0 & \sigma / \bar{z}_{\mathrm{cc}}^{[2,1]} & 0 \\ -\sigma / \bar{z}_{\mathrm{cc}}^{[2,1]} & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.23}$$

If (7.20) holds, then

$$\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \boldsymbol{R}\left(\begin{bmatrix} 0 \\ 0 \\ -\frac{\pi}{2} \end{bmatrix}\right) = \boldsymbol{R}\left(-\frac{\pi}{2} \boldsymbol{e}_3\right). \tag{7.24}$$

If (7.21) holds, then

$$\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \boldsymbol{R}\left(\begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{2} \end{bmatrix}\right) = \boldsymbol{R}\left(\frac{\pi}{2} \boldsymbol{e}_3\right). \tag{7.25}$$

In (7.24) and (7.25), $\boldsymbol{R}(\cdot)$ denotes the rotation function that maps a rotation vector to its associated rotation matrix via the method presented in Sect. 5.2.1 in Chap. 5. Once $\bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]}$ is obtained, apply the second equation of (7.18) to recover the rotation matrix $\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}$ as

$$\boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \bar{\boldsymbol{R}}_{\mathrm{cc}}^{[2,1]} \boldsymbol{V}^{\mathrm{T}}. \tag{7.26}$$

Binocular triangulation and distance positiveness verification

In summary, the solutions of (7.16)

$$\boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{E}$$

are

$$\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} = \sigma \boldsymbol{U} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \sigma \boldsymbol{U} \boldsymbol{e}_3, \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{V}^{\mathrm{T}} = \boldsymbol{U} \boldsymbol{R}\left(-\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}$$

or

$$\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} = \sigma \boldsymbol{U} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \sigma \boldsymbol{U} (-\boldsymbol{e}_3), \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{V}^{\mathrm{T}} = \boldsymbol{U} \boldsymbol{R}\left(\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}.$$

Considering the scaling factor (positive or negative), the solutions of (4.73)

$$\boldsymbol{S}_{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}} \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{E}$$

are

$$\begin{aligned}
&\text{solution 1}: \quad \boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{U} \boldsymbol{e}_3, \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \boldsymbol{R}\left(-\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}, \\
&\text{solution 2}: \quad \boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{U} (-\boldsymbol{e}_3), \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \boldsymbol{R}\left(-\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}, \\
&\text{solution 3}: \quad \boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{U} \boldsymbol{e}_3, \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \boldsymbol{R}\left(\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}, \\
&\text{solution 4}: \quad \boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{U} (-\boldsymbol{e}_3), \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \boldsymbol{R}\left(\tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}},
\end{aligned}$$

or expressed compactly as

$$\boldsymbol{T}_{\mathrm{cc}}^{[2,1]} \propto \boldsymbol{U} (\pm \boldsymbol{e}_3), \quad \boldsymbol{R}_{\mathrm{cc}}^{[2,1]} = \boldsymbol{U} \boldsymbol{R}\left(\pm \tfrac{\pi}{2} \boldsymbol{e}_3 \right) \boldsymbol{V}^{\mathrm{T}}. \tag{7.27}$$

However, not all the four solutions[5] are consistent with camera physical configurations in practice. In fact, only one of the four solutions satisfies such consistency and hence is valid. To determine which one is the valid solution, one can resort to *binocular triangulation* with a pair of matched feature points.

Given a pair of matched feature points with image coordinates denoted as $\boldsymbol{P}_{\mathrm{i}}^{[1]} \equiv \left[u^{[1]} \quad v^{[1]}\right]^{\mathrm{T}}$ and $\boldsymbol{P}_{\mathrm{i}}^{[2]} \equiv \left[u^{[2]} \quad v^{[2]}\right]^{\mathrm{T}}$ respectively. The following equalities hold

$$\lambda^{[1]}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} \equiv \lambda^{[1]}\begin{bmatrix}\alpha & \gamma & u_0\\ 0 & \beta & v_0\\ 0 & 0 & 1\end{bmatrix}^{-1}\begin{bmatrix}u^{[1]}\\ v^{[1]}\\ 1\end{bmatrix} = \begin{bmatrix}x_{\mathrm{c}}^{[1]}\\ y_{\mathrm{c}}^{[1]}\\ z_{\mathrm{c}}^{[1]}\end{bmatrix} \equiv \boldsymbol{P}_{\mathrm{c}}^{[1]}, \tag{7.28a}$$

$$\lambda^{[2]}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} \equiv \lambda^{[2]}\begin{bmatrix}\alpha & \gamma & u_0\\ 0 & \beta & v_0\\ 0 & 0 & 1\end{bmatrix}^{-1}\begin{bmatrix}u^{[2]}\\ v^{[2]}\\ 1\end{bmatrix} = \begin{bmatrix}x_{\mathrm{c}}^{[2]}\\ y_{\mathrm{c}}^{[2]}\\ z_{\mathrm{c}}^{[2]}\end{bmatrix} \equiv \boldsymbol{P}_{\mathrm{c}}^{[2]}, \tag{7.28b}$$

$$\boldsymbol{P}_{\mathrm{c}}^{[1]} = \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{P}_{\mathrm{c}}^{[2]} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]}. \tag{7.28c}$$

Substitute the first and second equations of (7.28) into the third equation of (7.28) and obtain

$$\begin{aligned}\lambda^{[1]}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} &= \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\lambda^{[2]}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A} + \boldsymbol{T}_{\mathrm{cc}}^{[2,1]}\\ \Leftrightarrow \quad \left[\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[1]A} \quad -\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\boldsymbol{A}^{-1}\boldsymbol{P}_{\mathrm{i}}^{[2]A}\right]\begin{bmatrix}\lambda^{[1]}\\ \lambda^{[2]}\end{bmatrix} &= \boldsymbol{T}_{\mathrm{cc}}^{[2,1]}.\end{aligned} \tag{7.29}$$

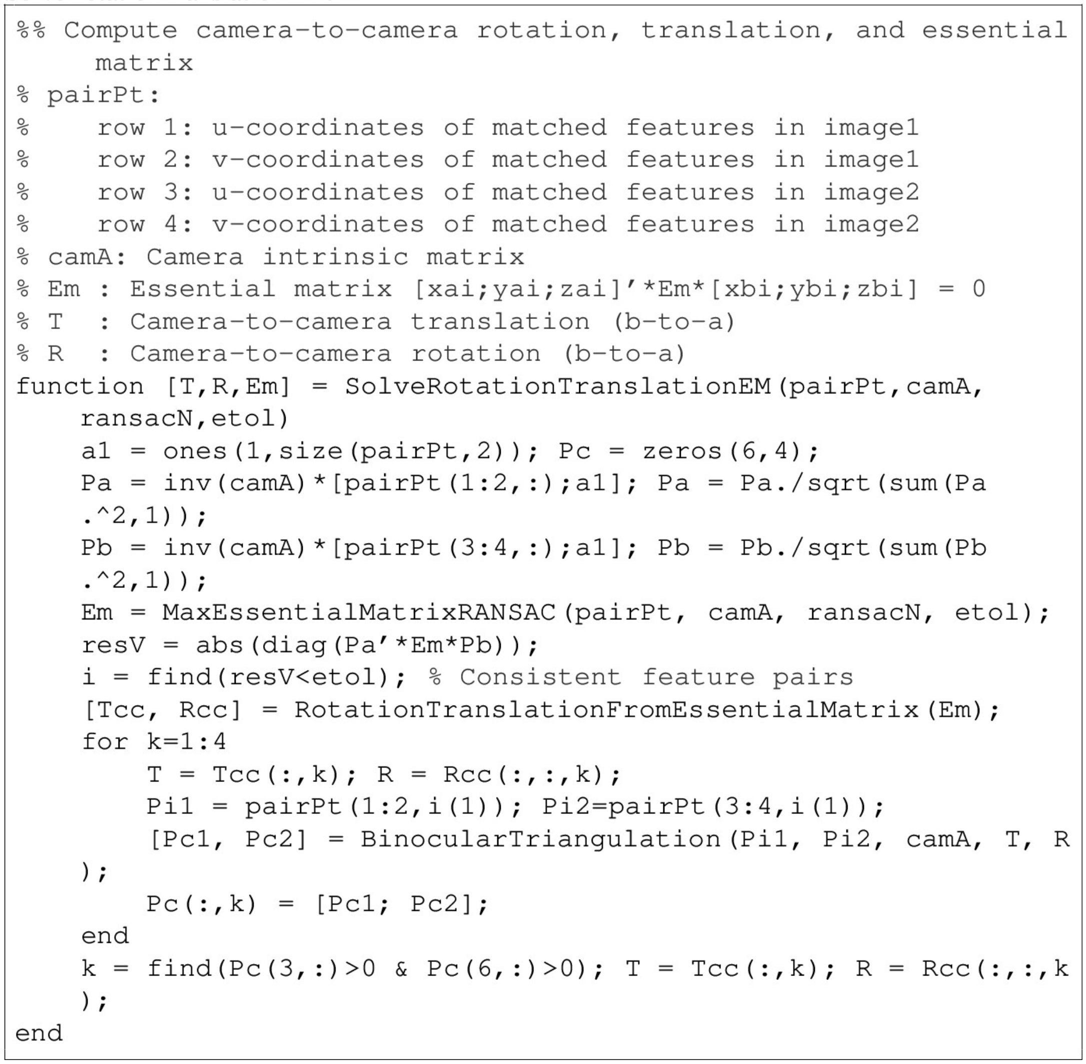
Solve the linear equation (7.29) via the method presented in Sect. F.1 in Appendix F. Once the solution of $\{\lambda^{[1]}, \lambda^{[2]}\}$ is obtained, the three-dimensional point that corresponds to the pair of matched feature points can be located in the camera coordinates systems via the first and second equations of (7.28). Matlab code for binocular triangulation with a pair of image points is given as follows.

BinocularTriangulation.m

```
%% Binocular triangulation
function [Pc1, Pc2] = BinocularTriangulation(Pi1, Pi2, camA, T21
    , R21)
    Pc1 = inv(camA)*[Pi1;1];
    Pc2 = inv(camA)*[Pi2;1];
    lambda = SolveLinearEquation([Pc1, -R21*Pc2], T21);
    Pc1 = lambda(1)*Pc1;
    Pc2 = lambda(2)*Pc2;
end
```

To find the valid one among the four solutions given in (7.27), one just needs to check which $\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]}, \boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}$ given in (7.27) enables the solution of $\{\lambda^{[1]}, \lambda^{[2]}\}$ to be both positive, because the three-dimensional point viewed from both camera poses must have positive distances, i.e. positive z_{c} coordinates in the camera coordinates systems. Matlab code for estimating the inter-camera relative pose is given as follows.

SolveRotationTranslationEM.m



```
%% Compute camera-to-camera rotation, translation, and essential
     matrix
% pairPt:
%    row 1: u-coordinates of matched features in image1
%    row 2: v-coordinates of matched features in image1
%    row 3: u-coordinates of matched features in image2
%    row 4: v-coordinates of matched features in image2
% camA: Camera intrinsic matrix
% Em : Essential matrix [xai;yai;zai]’*Em*[xbi;ybi;zbi] = 0
% T  : Camera-to-camera translation (b-to-a)
% R  : Camera-to-camera rotation (b-to-a)
function [T,R,Em] = SolveRotationTranslationEM(pairPt,camA,
    ransacN,etol)
    a1 = ones(1,size(pairPt,2)); Pc = zeros(6,4);
    Pa = inv(camA)*[pairPt(1:2,:);a1]; Pa = Pa./sqrt(sum(Pa
    .^2,1));
    Pb = inv(camA)*[pairPt(3:4,:);a1]; Pb = Pb./sqrt(sum(Pb
    .^2,1));
    Em = MaxEssentialMatrixRANSAC(pairPt, camA, ransacN, etol);
    resV = abs(diag(Pa’*Em*Pb));
    i = find(resV<etol); % Consistent feature pairs
    [Tcc, Rcc] = RotationTranslationFromEssentialMatrix(Em);
    for k=1:4
        T = Tcc(:,k); R = Rcc(:,:,k);
        Pi1 = pairPt(1:2,i(1)); Pi2=pairPt(3:4,i(1));
        [Pc1, Pc2] = BinocularTriangulation(Pi1, Pi2, camA, T, R
    );
        Pc(:,k) = [Pc1; Pc2];
    end
    k = find(Pc(3,:)>0 & Pc(6,:)>0); T = Tcc(:,k); R = Rcc(:,:,k
    );
end
```

The sub-function **MaxEssentialMatrixRANSAC.m** that applies the *RANdom SAmple Consensus* (RANSAC) method presented in Sect. 6.3.3 in Chap. 6 to estimate the essential matrix most consistent with a set of matched feature points and the sub-function **RotationTranslationFromEssentialMatrix.m** that solves the camera-to-camera translation and rotation candidates from a given essential matrix are given as follows. The sub-function **CalibrateEssentialMatrix.m** is already given in Sect. 5.4.2 in Chap.5, whereas the sub-function **RotationVectorToMatrix.m** is already given in Sect. 5.2.1 in Chap. 5.

MaxEssentialMatrixRANSAC.m

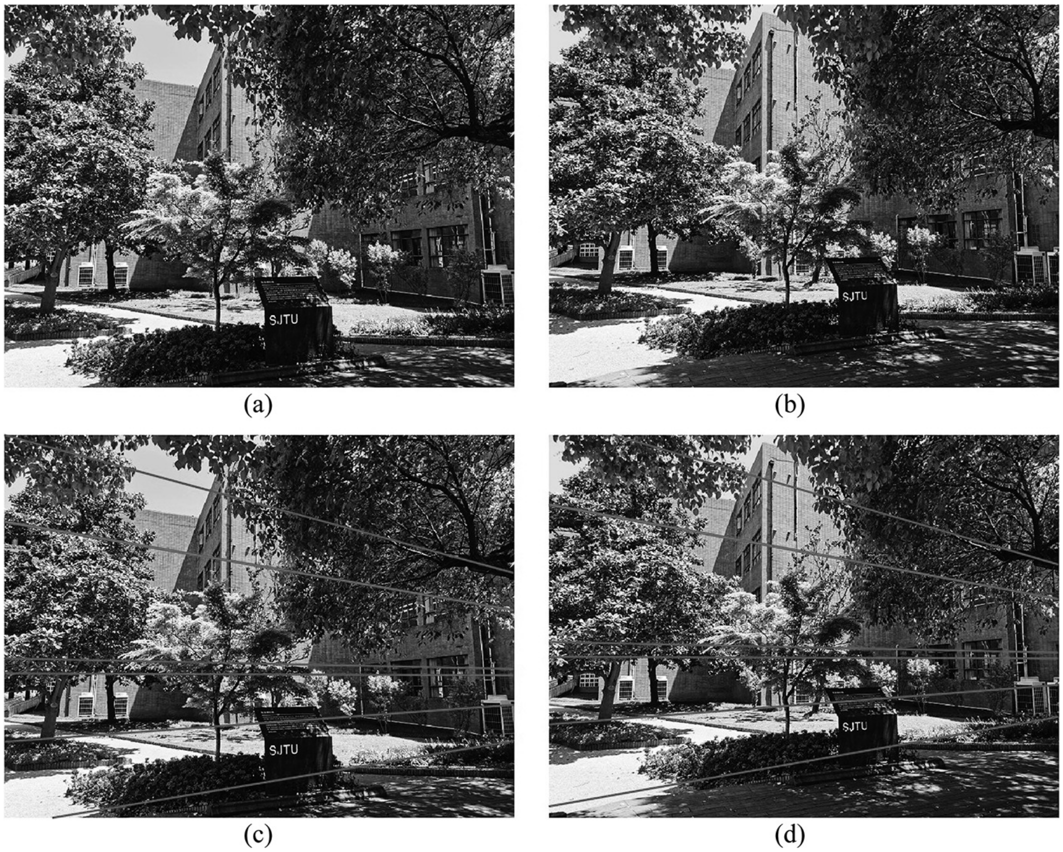
```
%% Maximize essential matrix consensus via RANSAC
% Compute essential matrix maximizing correspondence consensus
% i.e. consistent pairs between two images
% pairPt:
%    row 1: u-coordinates of matched features in image1
%    row 2: v-coordinates of matched features in image1
%    row 3: u-coordinates of matched features in image2
%    row 4: v-coordinates of matched features in image2
% Em: Essential matrix [xai;yai;zai]’*Em*[xbi;ybi;zbi] = 0
% camA: Camera intrinsic matrix
function Em_opt = MaxEssentialMatrixRANSAC(pairPt, camA, ransacN
    , etol)
    cn = 0; a1 = ones(1,size(pairPt,2));
    u1 = pairPt(1,:); v1 = pairPt(2,:); Pa = inv(camA)*[u1;v1;a1
    ];
    u2 = pairPt(3,:); v2 = pairPt(4,:); Pb = inv(camA)*[u2;v2;a1
    ];
    Pa = Pa./sqrt(sum(Pa.^2,1)); Pb = Pb./sqrt(sum(Pb.^2,1));
    for k = 1:ransacN
        % Random sampling
        i = ceil(rand(1,9)*size(pairPt,2));
        u1t=u1(i); v1t=v1(i); u2t=u2(i); v2t=v2(i);
        Em = CalibrateEssentialMatrix(u1t,v1t,u2t,v2t,camA);
        resV = abs(diag(Pa’*Em*Pb));
        % Consensus
        i = find(resV<etol);
        u1t=u1(i); v1t=v1(i); u2t=u2(i); v2t=v2(i); i = length(i
    );
        if (i>cn)
            fprintf(’%d-th RANSAC: %d good pairs\n’,k,i);
            cn = i;
            Em_opt = CalibrateEssentialMatrix(u1t,v1t,u2t,v2t,
    camA);
        end
    end
    fprintf(’FINAL (N=%d): %d good pairs \ n’,ransacN,cn);
end
```

RotationTranslationFromEssentialMatrix.m

```
%% Solve the rotation and translation from the essential matrix
% Tcc : Camera-to-camera translation candidates
% Rcc : Camera-to-camera rotation candidates
function [Tcc, Rcc] = RotationTranslationFromEssentialMatrix(Em)
    Tcc = zeros(3,4); Rcc = zeros(3,3,4); e3 = [0;0;1];
    [U,S,V] = svd(Em);
    if(det(U)<0) U = -U; end
    if(det(V)<0) V = -V; end
    for k=1:4
        sT = (-1)^k; sR = (-1)^floor((k-1)/2);
        Tcc(:,k) = U*(sT*e3);
        Rcc(:,:,k) = U*RotationVectorToMatrix(sR*(pi/2)*e3)*V’;
    end
end
```

Demonstration

An application example of two images captured consecutively by a camera is illustrated in Fig. 7.22a, b. The sub-figure(a) illustrates the image captured at the first camera pose, whereas the sub-figure(b) illustrates the image captured at the second camera pose.



(a) (b)

(c) (d)

Fig. 7.22 Visual SLAM **a** image captured at the first camera pose; **b** image captured at the second camera pose; **c** first image with epipolar lines; **d** second image with corresponding epipolar lines

Extract SIFT feature points from both the first and second images—Other kinds of feature points such as ORB can also be used—Match SIFT feature points between the two images according to their similarities. Then refine SIFT correspondences via the essential matrix based RANSAC method, namely to figure out the optimal essential matrix that maximizes consensus among SIFT correspondences and extract the SIFT correspondences consistent with the optimal essential matrix . Apply the sub-function **MaxEssentialMatrixRANSAC.m** given above to realize the intended RANSAC method. The refined SIFT correspondences are demonstrated in Fig. 7.23a. The estimated essential matrix is

$$\boldsymbol{E}=\begin{bmatrix}-0.000\,4 & 0.379\,5 & 0.028\,2\\ -0.403\,0 & 0.024\,9 & 0.578\,9\\ -0.045\,8 & -0.595\,4 & 0.022\,1\end{bmatrix}.$$

Once the essential matrix is available (and hence the fundamental matrix can also be derived), the epipolar constraint between the two images is determined. Some epipolar lines are demonstrated in Fig. 7.22c, d.

Then apply the sub-function **RotationTranslationFromEssentialMatrix.m** given above to perform singular value decomposition of $\boldsymbol{E}$ and then compute the camera-to-camera translation and rotation candidates accordingly. The four pairs of candidates with normalized translation vectors are

$$\begin{aligned}
\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]},\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}_{C_1} &= \left\{\begin{bmatrix}-0.842\,8\\ 0.061\,6\\ -0.534\,6\end{bmatrix},\begin{bmatrix}0.381\,8 & -0.068\,2 & 0.921\,7\\ -0.104\,7 & -0.994\,0 & -0.030\,2\\ 0.918\,3 & -0.085\,0 & -0.386\,6\end{bmatrix}\right\},\\
\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]},\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}_{C_2} &= \left\{\begin{bmatrix}0.842\,8\\ -0.061\,6\\ 0.534\,6\end{bmatrix},\begin{bmatrix}0.381\,8 & -0.068\,2 & 0.921\,7\\ -0.104\,7 & -0.994\,0 & -0.030\,2\\ 0.918\,3 & -0.085\,0 & -0.386\,6\end{bmatrix}\right\},\\
\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]},\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}_{C_3} &= \left\{\begin{bmatrix}-0.842\,8\\ 0.061\,6\\ -0.534\,6\end{bmatrix},\begin{bmatrix}0.999\,1 & -0.002\,2 & 0.042\,5\\ 0.003\,9 & 0.999\,2 & -0.040\,2\\ -0.042\,4 & 0.040\,4 & 0.998\,3\end{bmatrix}\right\},\\
\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]},\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}_{C_4} &= \left\{\begin{bmatrix}0.842\,8\\ -0.061\,6\\ 0.534\,6\end{bmatrix},\begin{bmatrix}0.999\,1 & -0.002\,2 & 0.042\,5\\ 0.003\,9 & 0.999\,2 & -0.040\,2\\ -0.042\,4 & 0.040\,4 & 0.998\,3\end{bmatrix}\right\}.
\end{aligned}$$

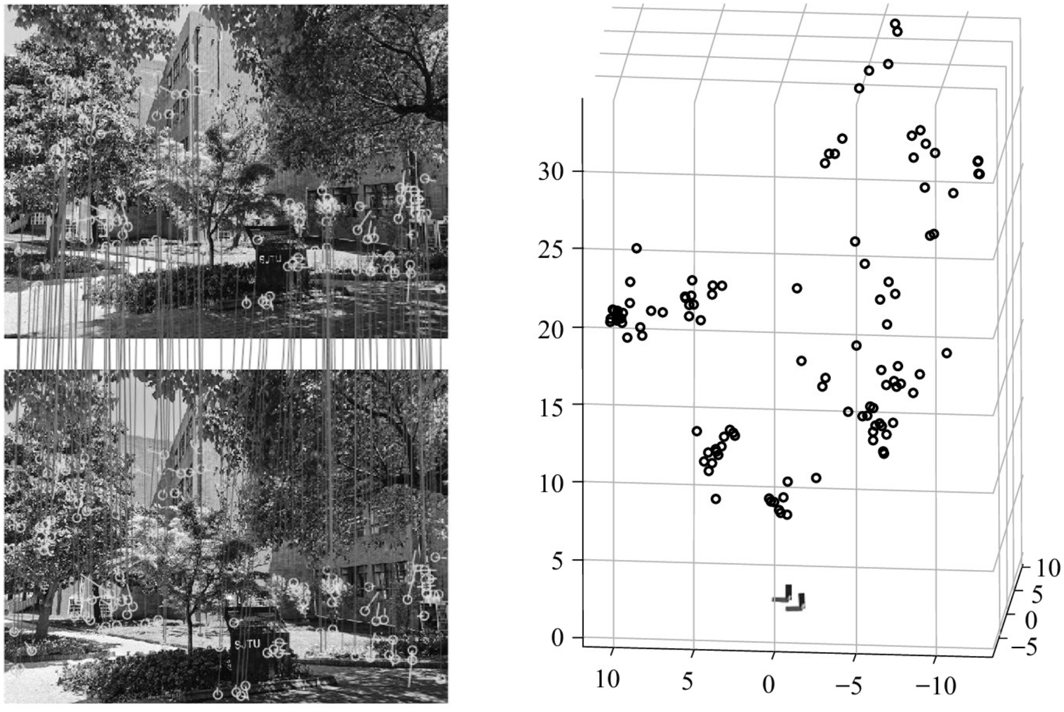
Substitute above four candidates of $\{\boldsymbol{T}_{\mathrm{cc}}^{[2,1]},\boldsymbol{R}_{\mathrm{cc}}^{[2,1]}\}$ into the linear equation (7.29) to solve their corresponding solutions of $\{\lambda^{[1]},\lambda^{[2]}\}$ respectively as

$$\begin{bmatrix}\lambda^{[1]}\\ \lambda^{[2]}\end{bmatrix}_{C_1}=\begin{bmatrix}-4.363\,2\\ 4.543\,6\end{bmatrix},\quad \begin{bmatrix}\lambda^{[1]}\\ \lambda^{[2]}\end{bmatrix}_{C_2}=\begin{bmatrix}4.363\,2\\ -4.543\,6\end{bmatrix},$$
$$\begin{bmatrix}\lambda^{[1]}\\ \lambda^{[2]}\end{bmatrix}_{C_3}=\begin{bmatrix}10.753\,7\\ 11.198\,2\end{bmatrix},\quad \begin{bmatrix}\lambda^{[1]}\\ \lambda^{[2]}\end{bmatrix}_{C_4}=\begin{bmatrix}-10.753\,7\\ -11.198\,2\end{bmatrix}.$$

The only translation and rotation candidate that enables the solution of $\{\lambda^{[1]},\lambda^{[2]}\}$ to be both positive is the third candidate, which is taken as the final solution of the inter-camera relative pose

$$\boldsymbol{T}_{\text{cc}}^{[2,1]}=\left\{\begin{bmatrix}-0.842\,8\\ 0.061\,6\\ -0.534\,6\end{bmatrix},\quad \boldsymbol{R}_{\text{cc}}^{[2,1]}=\begin{bmatrix}0.999\,1 & -0.002\,2 & 0.042\,5\\ 0.003\,9 & 0.999\,2 & -0.040\,2\\ -0.042\,4 & 0.040\,4 & 0.998\,3\end{bmatrix}.\right.$$

To visualize mapping of SIFT feature points according to the obtained solution of the inter-camera relative pose, apply the sub-function **BinocularTriangulation.m** to compute SIFT feature positions in the first camera coordinates system, as demonstrated in Fig. 7.23b. Besides, visualize depths of refined SIFT feature points via an augmented reality effect on the original camera image, as demonstrated in Fig. 7.24, where SIFT feature points are represented by circles, sizes of which are inverse proportional to their depths. The augmented reality effect is consistent with intuitive observation of the environment, which reflects validity of the estimated inter-camera relative pose.



(a) (b)

Fig. 7.23 Visual SLAM **a** refined SIFT correspondences; **b** binocular triangulated three-dimensional positions of SIFT feature points

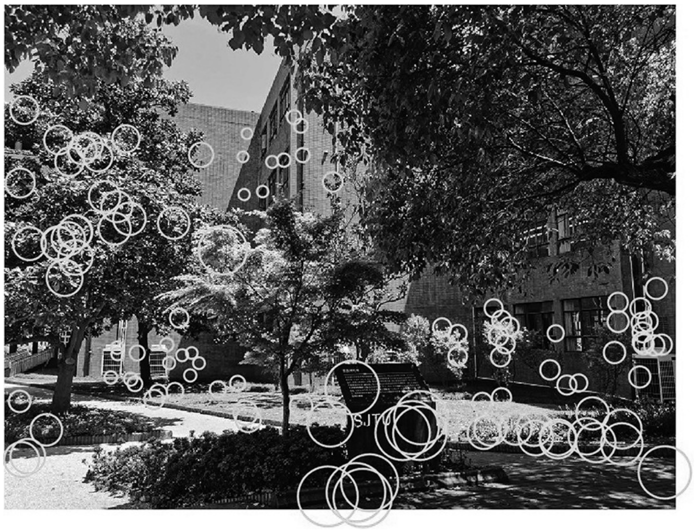


Fig. 7.24 Depths of refined SIFT feature points

References

1. H. Li, F. Nashashibi. *Lane detection (part i): Mono-vision based method*. INRIA Tech Report, RT-433 (2013)
2. M. Bertozzi, A. Broggi, A. Fascioli, Stereo inverse perspective mapping: theory and applications. Image Vis. Comput. **16**(8), 585–590 (1998)
[Crossref]

3. H. Li, F. Nashashibi, Multi-vehicle cooperative perception and augmented reality for driver assistance: a possibility to see through front vehicle, in *IEEE International Conference on Intelligent Transportation Systems* (2011), pp. 242–247
4. H. Li. *Cooperative Perception: Application in the Context of Outdoor Intelligent Vehicle Systems*. Ph.D. Thesis, Mines ParisTech (2012)
5. F. Rameau, H. Ha, K. Joo, J. Choi, K. Park, I.S. Kweon, A real-time augmented reality system to see-through cars. IEEE Trans. Vis. Comput. Graph. **22**(11), 2395–2404 (2016) [Crossref][PubMed]
6. H. Li, M. Tsukada, F. Nashashibi, M. Parent, Multivehicle cooperative local mapping: a methodology based on occupancy grid map merging. IEEE Trans. Intell. Transp. Syst. **15**(5), 2089–2100 (2014) [Crossref]
7. S. Fang, H. Li, Multi-vehicle cooperative simultaneous lidar slam and object tracking in dynamic environments. IEEE Trans. Intell. Transp. Syst. **25**(9), 11411–11421 (2024) [Crossref]
8. R. Hamming, Error detecting and error correcting codes. Bell Syst. Tech. J. **29**(2), 147–160 (1950) [ADS][MathSciNet][Crossref][zbMATH]
9. E. Olson, J. Strom, R. Morton, A. Richardson, P. Ranganathan, R. Goeddel, M. Bulic, J. Crossman, B. Marinier, Progress toward multi-robot reconnaissance and the magic 2010 competition. J. Field Robot. **29**(5), 762–792 (2012) [Crossref]
10. J. Wang, E. Olson, Apriltag 2: efficient and robust fiducial detection, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2016), pp. 4193–4198
11. S. Fang, Y. Li, H. Li, Split covariance intersection filter based visual localization with accurate apriltag map for warehouse robot navigation (2023)
12. A. Davison, I. Reid, N. Molton, O. Stasse, Monoslam: real-time single camera slam. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 1052–1067 (2007) [ADS][Crossref][PubMed]
13. G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in *IEEE and ACM International Symposium on Mixed and Augmented Reality* (2007), pp. 225–234
14. R. Mur-Artal, J. Montiel, J. Tardós, Orb-slam: a versatile and accurate monocular slam system. IEEE Trans. Robot. **31**(5), 1147–1163 (2015) [Crossref]

Footnotes

[1] Nowadays, the 5G and 5G+ communication technology can support real-time communication of a large amount of perception data.

[2] The logic actually does not apply when a special coincidence is encountered, namely the edge cluster happens to be a simple edge curve segment and the starting edge point happens to be at one end of the edge curve segment as well. For such seemingly “bug” of the logic, it is not difficult to remedy it, yet the author intentionally neglects it for two reasons: First, the special coincidence is hardly encountered in practice. Second, even if the special coincidence is indeed encountered, the wrongly detected simple cycle, i.e. a “false positive” will after all be sifted out in following steps and cause no harm.

[3] The reference [11] is a technical report of the team directed by the author.

[4] The inter-camera relative pose refers to the relative pose between the two camera poses or the two camera coordinates systems associated with the pair of images respectively.

[5] Each solution represents actually a family of solutions that differ from each other only by a scaling factor.

Appendix A

Integral Image

Given an image and the image pixel value function denoted as $I(u, v)$ ($1 \leqslant u \leqslant W$, $1 \leqslant v \leqslant H$); W and H denote the image width and height respectively.

The integral image function $I_S(u, v)$ is defined in the discrete $[u, v]^{\mathrm{T}}$ domain as

$$I_S(u, v) = \sum_{r=1}^{v} \sum_{c=1}^{u} I(c, r),$$

where $u \in \{1, \cdots, W\}$ and $v \in \{1, \cdots, H\}$ are integers. The integral image function $I_S(u, v)$ is defined generally in the continuous $[u, v]^{\mathrm{T}}$ domain as

$$I_S(u, v) = \int\limits_{1}^{v} \int\limits_{1}^{u} I(u, v) \mathrm{d}u \mathrm{d}v \tag{A.1}$$

or for short as

$$I_S(u, v) = \int\limits_{1,1}^{u,v} I(u, v) \mathrm{d}u \mathrm{d}v \equiv \int^{u,v} I(u, v) \mathrm{d}u \mathrm{d}v,$$

where $1 \leqslant u \leqslant W$ and $1 \leqslant v \leqslant H$. Besides, denote

$$\int\limits_{u_1,v_1}^{u_2,v_2} I(u, v) \mathrm{d}u \mathrm{d}v \equiv \int\limits_{v_1}^{v_2} \int\limits_{u_1}^{u_2} I(u, v) \mathrm{d}u \mathrm{d}v.$$

As illustrated in Fig. A.1, the integral image function $I_S(u, v)$ possesses the following property that facilitates computation of it:

$$I_S(u_2, v_2) + I_S(u_1, v_1) = I_S(u_2, v_1) + I_S(u_1, v_2) + \int\limits_{u_1,v_1}^{u_2,v_2} I(u, v) \mathrm{d}u \mathrm{d}v. \tag{A.2}$$

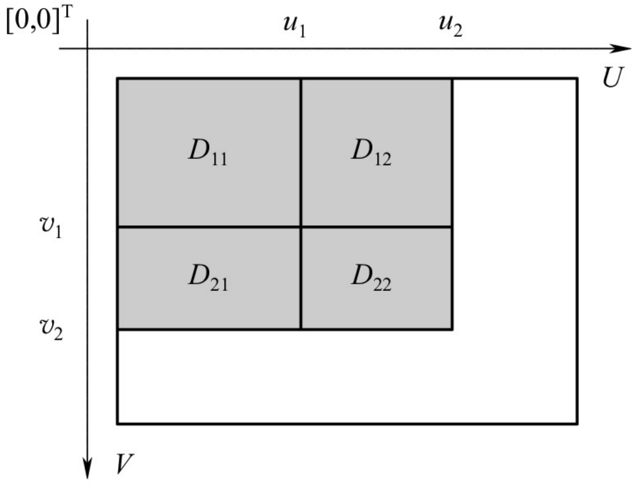


Fig. A.1 Integral image computation

Proof On one hand, one has

$$\begin{aligned} I_S(u_2,v_2)+I_S(u_1,v_1) &= \int\limits_{D_{11}+D_{12}+D_{21}+D_{22}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{11}} I(u,v)\mathrm{d}u\mathrm{d}v \\ &= 2\int\limits_{D_{11}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{12}} I(u,v)\mathrm{d}u\mathrm{d}v \\ &\quad + \int\limits_{D_{21}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{22}} I(u,v)\mathrm{d}u\mathrm{d}v. \end{aligned}$$

On the other hand, one has

$$\begin{aligned} & I_S(u_2,v_1)+I_S(u_1,v_2)+\int\limits_{u_1,v_1}^{u_2,v_2} I(u,v)\mathrm{d}u\mathrm{d}v \\ = & \int\limits_{D_{11}+D_{12}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{11}+D_{21}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{22}} I(u,v)\mathrm{d}u\mathrm{d}v \\ = & \ 2\int\limits_{D_{11}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{12}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{21}} I(u,v)\mathrm{d}u\mathrm{d}v + \int\limits_{D_{22}} I(u,v)\mathrm{d}u\mathrm{d}v. \end{aligned}$$

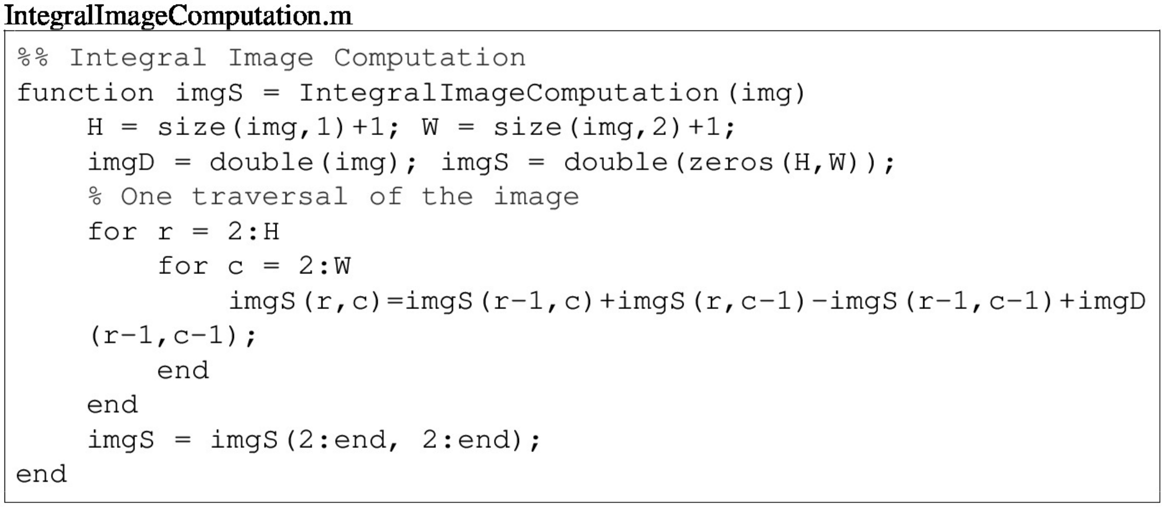
So (A.2) holds. The proof is done.

The property (A.2) leads naturally to the *integral image iterative computation* technique:

$$I_S(u,v)=I_S(u,v-1)+I_S(u-1,v)-I_S(u-1,v-1)+I(u,v), \quad \text{(A.3)}$$

which implies that the integral image can be computed just by one traversal of the image.

Matlab code for integral image computation is give as follows.



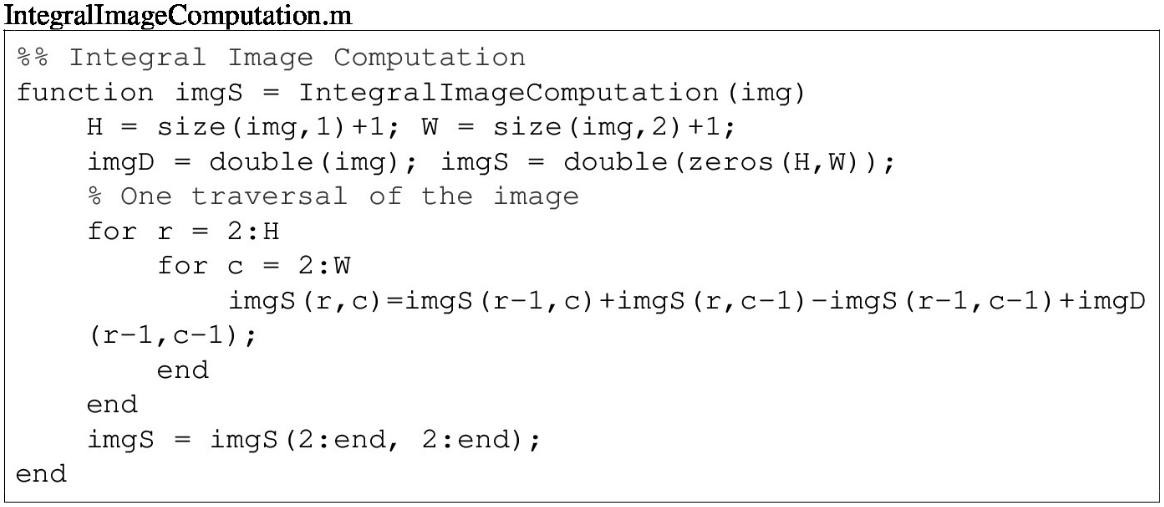
IntegralImageComputation.m

```
%% Integral Image Computation
function imgS = IntegralImageComputation(img)
    H = size(img,1)+1; W = size(img,2)+1;
    imgD = double(img); imgS = double(zeros(H,W));
    % One traversal of the image
    for r = 2:H
        for c = 2:W
            imgS(r,c)=imgS(r-1,c)+imgS(r,c-1)-imgS(r-1,c-1)+imgD
    (r-1,c-1);
        end
    end
    imgS = imgS(2:end, 2:end);
end
```

Once the integral image I_S is computed efficiently in above way. The integral image over any rectangular image area can also be obtained efficiently at a computational cost always of order $O(1)$, instead of order $O(\textit{image area size})$ for naive implementation. The efficient way of computing the integral image over a generic rectangular area $D_{u_1,v_1}^{u_2,v_2} \equiv \{[u,v]^{\mathrm{T}} \mid u_1 \leqslant u \leqslant u_2, v_1 \leqslant v \leqslant v_2\}$ is

$$\int\limits_{D_{u_1,v_1}^{u_2,v_2}} I(u,v)\mathrm{d}u\mathrm{d}v = I_S(u_2,v_2) - I_S(u_2,v_1) - I_S(u_1,v_2) + I_S(u_1,v_1). \tag{A.4}$$

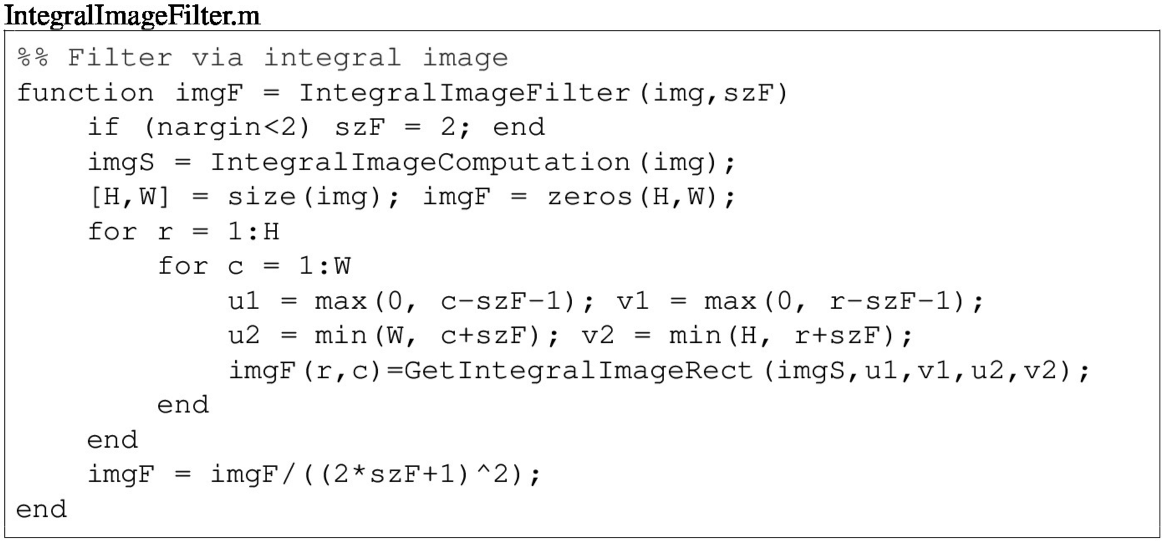
Matlab code for computing the integral image over a generic rectangular area is give as follows.



IntegralImageComputation.m

```
%% Integral Image Computation
function imgS = IntegralImageComputation(img)
    H = size(img,1)+1; W = size(img,2)+1;
    imgD = double(img); imgS = double(zeros(H,W));
    % One traversal of the image
    for r = 2:H
        for c = 2:W
            imgS(r,c)=imgS(r-1,c)+imgS(r,c-1)-imgS(r-1,c-1)+imgD
    (r-1,c-1);
        end
    end
    imgS = imgS(2:end, 2:end);
end
```

The integral image enables an efficient implementation of the image filter that smooths each pixel by averaging pixel values in the pixel’s surrounding local patch. Matlab code for the integral image based filter is given as follows. Such integral image based filter is also known as the box filter .



IntegralImageFilter.m

```
%% Filter via integral image
function imgF = IntegralImageFilter(img,szF)
    if (nargin<2) szF = 2; end
    imgS = IntegralImageComputation(img);
    [H,W] = size(img); imgF = zeros(H,W);
    for r = 1:H
        for c = 1:W
            u1 = max(0, c-szF-1); v1 = max(0, r-szF-1);
            u2 = min(W, c+szF); v2 = min(H, r+szF);
            imgF(r,c)=GetIntegralImageRect(imgS,u1,v1,u2,v2);
        end
    end
    imgF = imgF/((2*szF+1)^2);
end
```

Code Demonstration Note

As mentioned in Sect. 1.5 in Chap. 1, the author tends to incline towards the *practicalness* concern if the *conciseness* concern is not sacrificed too much. This somehow implies the opposite preference as well: the author tends to incline towards the conciseness concern if it would otherwise be indeed sacrificed too much whereas there is no gain at the complexity order level. Some explanations hover over such opposite preference, taking above Matlab scripts as example.

The Matlab script **IntegralImageComputation.m** is by no means optimal in terms of the practicalness concern. In the demonstrated code, with the help of an auxiliary matrix that augments the size of the original image along both dimensions by one, considerations of various boundary cases in the loop are unified into one simple logic for sake of code conciseness. Usage of the auxiliary matrix incurs extra computational cost namely one more traversal of the image to copy the integral image back from the auxiliary matrix, which is comparatively not cheap in the discussed context because the integral image iterative computation itself takes just one traversal of the image. The extra computational cost can fairly be saved, only if the loop is decomposed into multiple blocks according to different boundary cases, the pseudo code of which is given below.

Algorithm: Integral Image Computation

```
UPDATE first pixel OF first row OF imgS
FOR EACH pixel FROM second pixel OF first row
    UPDATE pixel
END
FOR EACH row FROM second row OF imgS
    UPDATE first pixel OF row
    FOR EACH pixel FROM second pixel OF row
        UPDATE pixel
    END
END
```

Compared with the original code logic in the Matlab script **IntegralImageComputation.m**, the code logic in above pseudo code becomes “swollen” apparently and sacrifices the conciseness concern too much, though it does favour the practicalness concern. Throughout this book, the author tends to give concise code that implements the kind of simple logic in the Matlab script **IntegralImageComputation.m** instead of complex code that implements the kind of “swollen” logic in the pseudo code.

In terms of the practicalness concern, the Matlab script **IntegralImageFilter.m** can also be improved if “c-szF-1”, “r-szF-1”, “c+szF”, “r+szF” in the loop are set as four independent variables that iterate with the loop. Throughout this book, however, the author also refrains from such kind of trivial improvement that will make code “swollen” but have no gain at the complexity order level.

Appendix B

Illusion of Brightness

Different objects under certain illumination condition have their different levels of physical brightness that can be captured and encoded as gray-level images. However, humans' subjective perception of an object's brightness does not depend only on the object's physical brightness. In fact, humans' subjective perception of an object's brightness also depends on the brightness contrast between the object and its surrounding objects.

This is due to humans' biological vision mechanism that strengthens brightness or intensity contrast. An object seems to be even brighter than its physical brightness if it is physically brighter than its surrounding objects, whereas an object seems to be even darker than its physical brightness if it is physically darker than its surrounding objects. In other words, humans tend to "exaggerate" an object's brightness when there is physical brightness contrast between the object and its surrounding objects. Besides, the larger the physical brightness contrast is, the more the "exaggeration" is.

For example, take a look at the two large squares and their respective central squares (namely the smaller squares embedded in them) illustrated in Fig. B.1. Which central square is brighter or darker? The author believes that readers would say the central square on the left is obviously brighter than that on the right, or in the opposite way, the right one is obviously darker than the left one.

For another example, take a look at the central band illustrated in Fig. B.2. Which side of the central band is brighter or darker? The author believes that readers would say the left side of the central band is obviously brighter than the right side, or in the opposite way, the right side is obviously darker than the left side.

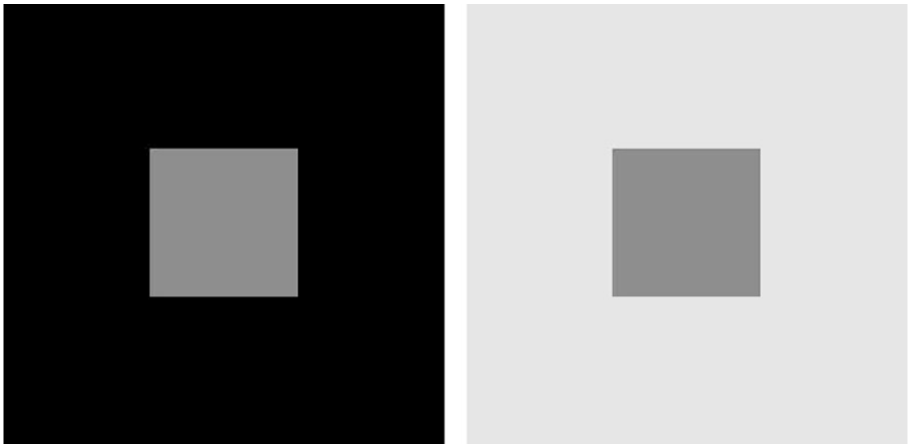


Fig. B.1 Illusion of brightness: which central square is brighter or darker?

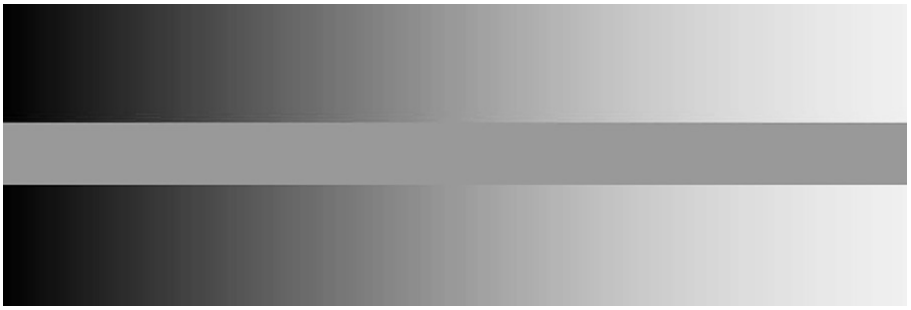


Fig. B.2 Illusion of brightness: which side of the central band is brighter or darker?

However, the two central squares illustrated in Fig. B.1 have the same physical brightness, and the entire central band illustrated in Fig. B.2 has even physical brightness. In other words, the left central square illustrated in Fig. B.1 is physically as bright as that on the right. The left side of the central band illustrated in Fig. B.2 is physically as bright as the right side—To recognize this point, readers can use paper to hide surrounding areas of the two central squares in Fig. B.1 and would then find that they actually have the same brightness. Readers can also use paper to hide surrounding areas of the central band in Fig. B.2 and would then find that the central band is actually of even brightness.

Humans’ illusions reflected by these two examples can be well explained by the reasons given above. In Fig. B.1, the central square on the left is surrounded by physically darker areas, so it seems brighter than its physical brightness. In contrast, the central square on the right is surrounded by physically brighter areas, so it seems darker than its physical brightness. Then a simple logic namely

$$\begin{aligned}
& \mathrm{left\,central\,square} * \mathrm{rsquo} * \mathrm{s\,seemingly\,brightness} \\
> \; & \mathrm{left\,central\,square} * \mathrm{rsquo} * \mathrm{s\,physical\,brightness} \\
= \; & \mathrm{right\,central\,square} * \mathrm{rsquo} * \mathrm{s\,physical\,brightness} \\
> \; & \mathrm{right\,central\,square} * \mathrm{rsquo} * \mathrm{s\,seemingly\,brightness}
\end{aligned}$$

can account for the illusion reflected by Fig. B.1. Explanations for the illusion reflected by Fig. B.2 can be given similarly and omitted here.

Appendix C

Image Filtering

An image can be regarded as a kind of two-dimensional signals, and image filtering can hence be treated as a special kind of **signal filtering**. To clarify fundamental principles of signal filtering, one can fairly focus on analysis dedicated to one-dimensional signal filtering, after which the principles can be naturally extended from one-dimensional signal filtering to two-dimensional signal filtering (including image filtering).

C.1 Basics of Probability and Stochastic Processes

Some basics of probability and stochastic processes [1] are reviewed.

C.1.1 Random Variables

Many quantities of interest encountered in practical applications are not deterministic,[1] but may potentially be determined in different ways with different probabilities. A typical example is the dice value obtained by dice tossing, which may potentially be one to six, with the probability of each of the six potential values to be one sixth. Such non-deterministic quantities are **random variables**. The set of potential values for a random variable is called the **value space** or **sample space** of the random variable.

If a random variable's value space is countable ,[2] it is said to be discrete. If its value space is uncountable, then it is a continuous random variable. For a discrete random variable x with finite value space $\{x\} = \{x_1, x_2, \cdots, x_n\}$ or infinite value space $\{x\} = \{x_1, x_2, \cdots\}$, its **probability mass function** $p(x)$ is defined as

$$p(x) = P\{x = x_i\}, \quad (x_i \in \{x\}), \tag{C.1}$$

where $P\{x = x_i\}$ denotes the probability of x being the potential value x_i ($x_i \in \{x\}$). The probability mass function $p(x)$ satisfies the non-negative condition

$$p(x) \geqslant 0$$

and the normalization condition

$$\textstyle\sum_{x \in \{x\}} p(x) = 1.$$

For a continuous random variable x with uncountable value space $\{x\}$, its **probability density function** $p(x)$ is defined in the way such that

$$P\{x \in S\} = \int\limits_S p(x) \mathrm{d}x$$

holds for an arbitrary subset S of $\{x\}$ (including the entire value space $\{x\}$), namely

$$P\{x \in S\} = \int\limits_S p(x) \mathrm{d}x, \quad (\forall S \subset \{x\}), \tag{C.2}$$

where $P\{x \in S\}$ denotes the probability of x belonging to the value set S. The probability density function $p(x)$ satisfies the non-negative condition

$$p(x) \geqslant 0 \tag{C.3}$$

and the normalization condition

$$\int_{\{x\}} p(x)\mathrm{d}x = 1. \tag{C.4}$$

For expression convenience, the author follows formalisms of continuous random variables throughout following analysis, be quantities of interest continuous ordiscrete—Formalisms of continuous random variables can be easily adapted for discrete cases. For example, the **cumulative distribution function** $F(x)$ associated with a probability density function $p(x)$ is defined as

$$F(x) \equiv P\{\tau \leqslant x\} = \int_{-\infty}^{x} p(\tau)\mathrm{d}\tau. \tag{C.5}$$

The formalism can be easily adapted to define the cumulative distribution function $F(x)$ associated with a probability mass function $p(x)$ as

$$F(x) \equiv P\{\tau \leqslant x\} = \textstyle\sum_{\tau \leqslant x} p(\tau).$$

Given a random variable x with value space $\{x\}$, its **expectation** or **expected value** $E[x]$ is defined as

$$E[x] = \int_{\{x\}} xp(x)\mathrm{d}x \quad \text{or} \quad \int_{-\infty}^{+\infty} xp(x)\mathrm{d}x, \tag{C.6}$$

where the value space $\{x\}$ can be naturally extended to $(-\infty, +\infty)$, satisfying

$$p(x) = \begin{cases} p(x), & x \in \{x\} \\ 0, & x \in (-\infty, +\infty)/\{x\} \end{cases} \tag{C.7}$$

The n-th **moment** of the random variable x is defined as

$$E[x^n] = \int_{\{x\}} x^n p(x)\mathrm{d}x \quad \text{or} \quad \int_{-\infty}^{+\infty} x^n p(x)\mathrm{d}x, \tag{C.8}$$

where the value space $\{x\}$ can be extended, as specified in (C.7). A random variable's expectation is its first moment.

A random variable x's offset random variable is defined as $\Delta x \equiv x - E[x]$, the first moment namely expectation of which is always

$$E[\Delta x] = E[x - E[x]] = E[x] - E[x] = 0.$$

The random variable x's **variance** $\mathrm{Var}(x)$ is defined as the second moment of the offset random variable Δx, namely

$$\mathrm{Var}(x) = E[\Delta x^2] = E[(x - E[x])^2] = E[x^2] - E[x]^2. \tag{C.9}$$

Given two random variables x and y, their **covariance** $\mathrm{Cov}(x, y)$ is defined as

$$\mathrm{Cov}(x,y) = E[\Delta x \Delta y] = E[(x - E[x])(y - E[y])] = E[xy] - E[x]E[y]. \tag{C.10}$$

If $y = x$, then $\mathrm{Cov}(x, y) = \mathrm{Var}(x)$. In other words, a random variable's variance can be regarded as a special kind of covariance, namely the covariance between the random variable and itself. If x and y are independent, then $\mathrm{Cov}(x, y) = 0$.

C.1.2 Limit Theorems

A number of useful theorems in probability theory are reviewed.

Markov inequality: Given a random variable x with value space $\{x\} \subset \mathbb{R}/\mathbb{R}^-$ (namely x takes only non-negative values), then for any value $a > 0$,

$$P\{x \geqslant a\} \leqslant \tfrac{E[x]}{a}.$$

Chebyshev inequality: Given a random variable x, then for any value $a > 0$,

$$P\{|x - E[x]| \geqslant a\} \leqslant \tfrac{\mathrm{Var}(x)}{a^2}.$$

When a random variable's probability distribution is known, one can compute the random variable's probabilities exactly according to its probability distribution. When a random variable's expectation and variance are known but its actual probability distribution is unknown, the Markov and Chebyshev inequalities at least enable one to infer bounds on the random variable's probabilities only according to its expectation and variance.

Strong law of large numbers: Given a sequence of *independent random variables* $x_1, x_2, \cdots$ that have a common probability distribution (and hence have a common expectation E[x]), then

$$\tfrac{x_1 + x_2 + \cdots + x_n}{n} \to E[x] \quad \text{as } n \to \infty$$

holds with probability 1.

Central limit theorem: Given a sequence of *independent random variables* $x_1, x_2, \cdots$ that have a common probability distribution (and hence have a common expectation E[x] and a common variance $\mathrm{Var}(x)$), then the probability distribution of

$$\tfrac{x_1 + x_2 + \cdots + x_n - nE[x]}{\sqrt{n \mathrm{Var}(x)}} \quad \text{as } n \to \infty$$

tends to be the *standard normal* or *standard Gaussian* distribution, namely

$$P\left\{ \tfrac{x_1 + x_2 + \cdots + x_n - nE[x]}{\sqrt{n \mathrm{Var}(x)}} \leqslant a \right\} \to \tfrac{1}{\sqrt{2\pi}} \int\limits_{-\infty}^{a} \mathrm{e}^{-x^2/2} \mathrm{d}x \quad \text{as } n \to \infty.$$

The importance and power of the *strong law of large numbers* and the *central limit theorem* consist in their general applicability, because they hold for *any probability distribution* of x_i $(i = 1, 2, \cdots)$.

C.1.3 Stochastic Processes

A **stochastic process** $\{x(t), t \in \{t\}\}$ is a collection of random variables, where $\{t\}$ is called the **index set** of the stochastic process. The term "process" tends to imply that a stochastic process usually describes certain system's evolution through time. Sometimes, one may also imagine a set of stationary elements as a stochastic process, as if the elements are generated consecutively through time. For example, an image can be regarded as two-dimensional signals that form a stochastic process.

Given a stochastic process $\{x(t), t \in \{t\}\}$, its **auto-correlation** at index t with index shift τ, denoted $A_x(t, \tau)$, is defined as the expectation of the product of the random variables $x(t)$ and $x(t+\tau)$, namely

$$A_x(t, \tau) = E[x(t)x(t+\tau)], \tag{C.11}$$

which can be treated as a bivariate function in terms of t and τ.

If $A_x(t, \tau)$ depends only on index shift τ regardless of index t, namely

$$A_x(t, \tau) = E[x(t)x(t+\tau)] \equiv A_x(\tau), \quad \forall t \in \{t\}, \tag{C.12}$$

then the stochastic process $\{x(t), t \in \{t\}\}$ is said to be **shift-invariant**. Besides, when a shift-invariant stochastic process $\{x(t), t \in \{t\}\}$ consists of a large number of random variables that have a common probability distribution, then according to the *strong law of large numbers*, the auto-correlation $A_x(\tau)$ can be fairly approximated by the integral

$$A_x(\tau) \approx \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} x(t)x(t+\tau)\mathrm{d}t. \tag{C.13}$$

Above concepts reviewed throughout Sect. C.1, which are defined for one-dimensional random variables, are also applicable to multiple-dimensional random variables. Formalisms that define the concepts for one-dimensional random variables can be easily adapted to define the same concepts for multiple-dimensional random variables, and details of such adaptation are omitted here.

C.2 Signal-Noise Modelling

For a signal-processing system, if the input sequence of signals is exempt from noises, then it is obviously unnecessary for the signal-processing system to perform filtering on the signal input. However, in practical applications, signals normally suffer from noises. What the signal-processing system actually takes as input is noisy signal input that is the true signal input overlapped by noises, as illustrated in Fig. C.1.

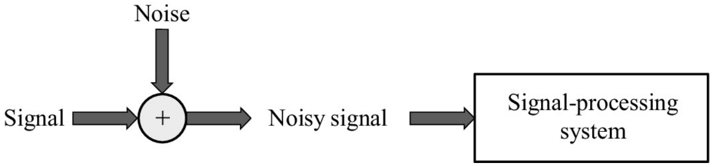


Fig. C.1 Signal-noise modelling

Denote the true input sequence of signals as $\{s(t), t \in \{t\}\}$ and the sequence of noises overlapping on the signal input as $\{n(t), t \in \{t\}\}$. Denote the sequence of noisy signals as $\{r(t), t \in \{t\}\}$:

$$r(t) = s(t) + n(t), \quad t \in \{t\}. \tag{C.14}$$

Intuitively speaking, the true signal input $\{s(t), t \in \{t\}\}$ seems to have no relation with any random factor, yet $\{s(t), t \in \{t\}\}$ may still be modelled as a stochastic process, as if each instant signal $s(t)$ is generated randomly according to a common probability distribution. Then the concepts presented in Sect. C.1 also apply to the signal input

$\{s(t), t \in \{t\}\}$. For example, its expectation, variance, and auto-correlation can be computed as[3]

$$\begin{aligned} E[s] &= \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} s(t)\mathrm{d}t, \\ \mathrm{Var}(s) &= \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} (s(t) - E[s])^2 \mathrm{d}t, \\ A_s(\tau) &= \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} s(t)s(t+\tau)\mathrm{d}t. \end{aligned} \tag{C.15}$$

In (C.15), $s(t+\tau)$ is defined according to a periodically extended version of $\{s(t), t \in \{t\}\}$, satisfying

$$s(t + k|\{t\}|) \equiv s(t), \quad \forall t \in \{t\}, k = \pm 1, \pm 2, \cdots.$$

The noise sequence $\{n(t), t \in \{t\}\}$ can also be modelled as a stochastic process. A common way is to model $\{n(t), t \in \{t\}\}$ as **white noises** [2], satisfying the following white noise conditions (WNC):

- *WNC 1*: $\forall t_1, t_2 \in \{t\}, t_1 \neq t_2$, $n(t_1)$ and $n(t_2)$ are mutually independent. In other words, $\{n(t), t \in \{t\}\}$ is a stochastic process consisting of independent random variables.
- *WNC 2*: $\forall t_1, t_2 \in \{t\}, t_1 \leqslant t_2$, the integral $\int_{t_1}^{t_2} n(t)\mathrm{d}t$ has the expectation 0 and the variance $n_0^2(t_2 - t_1)$, namely

  $$\begin{aligned} E\left[\int\limits_{t_1}^{t_2} n(t)\mathrm{d}t\right] &= 0, \\ \mathrm{Var}\left(\int\limits_{t_1}^{t_2} n(t)\mathrm{d}t\right) &= n_0^2(t_2 - t_1), \end{aligned}$$

  where the constant n_0^2 represents the energy over unit index length. Consider a generic infinitesimal integral interval $[t, t + \mathrm{d}t]$, and obtain a special limit case of *WNC 2*:

  $$\begin{aligned} E[n(t)\mathrm{d}t] &= E\left[\int\limits_{t}^{t+\mathrm{d}t} n(t)\mathrm{d}t\right] = 0, \\ \mathrm{Var}(n(t)\mathrm{d}t) &= \mathrm{Var}\left(\int\limits_{t}^{t+\mathrm{d}t} n(t)\mathrm{d}t\right) = n_0^2\mathrm{d}t. \end{aligned}$$

Consider the auto-correlation of $\{n(t), t \in \{t\}\}$. If $\tau \neq 0$, then according to*WNC 1*, one has

$$A_n(t, \tau) = 0.$$

If $\tau = 0$, then according to the special limit case of *WNC 2*, one has

$$A_n(t, \tau) = E[n(t)^2] = \mathrm{Var}(n(t)) = \tfrac{\mathrm{Var}(n(t)\mathrm{d}\tau)}{\mathrm{d}\tau^2} = \tfrac{n_0^2 \mathrm{d}\tau}{\mathrm{d}\tau^2} = \tfrac{n_0^2}{\mathrm{d}\tau}$$

and hence $A_n(t, \tau)\mathrm{d}\tau = n_0^2$. Therefore, $\{n(t), t \in \{t\}\}$ is a shift-invariant stochastic process with the auto-correlation function

$$A_n(t,\tau) = n_0^2\delta(\tau), \tag{C.16}$$

where $\delta(\tau)$ denotes the *Dirac function*.

It is worth noting that white noises are closely related to Brownian motion [2]. Define the function

$$B(t) = \int_0^t n(t)\mathrm{d}t, \tag{C.17}$$

which right describes a trajectory of Brownian motion. Besides, the function $B(t)$ possesses the interesting property that it is continuous everywhere but is nowhere differentiable.

C.3 Wiener Filter: Optimal Linear Filter

C.3.1 Filtering via Convolution

The signal-noise model is illustrated in Fig. C.1 and formalized in (C.14). What the signal-processing system actually takes as input is the noisy signal input

$$r(t) = s(t) + n(t), \quad t \in \{t\},$$

where $\{s(t), t \in \{t\}\}$ denotes the true input sequence of signals and $\{n(t), t \in \{t\}\}$ denotes the sequence of noises overlapping on the signal input.

For the signal-processing system, a preliminary step of filtering aims at revealing the true signal input $\{s(t)\}$ from the noisy signal input $\{r(t)\}$. In other words, from the perspective of signal flow, filtering aims at passing the true signal input $\{s(t)\}$ as much as possible while suppressing the noise part $\{n(t)\}$ as much as possible such that the filter output or the obtained signal estimates $\{\hat{s}(t)\}$ are as close as possible to the true signal input $\{s(t)\}$. The signal estimates $\{\hat{s}(t)\}$ which are better than the noisy signal input $\{r(t)\}$ are then further processed by the signal-processing system, as illustrated in Fig. C.2.

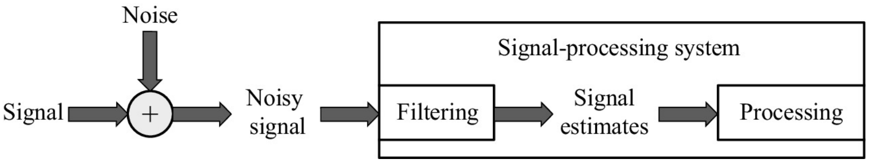


Fig. C.2 Signal-noise filtering

If the filter module is assumed to be a *linear time-invariant* dynamic system [3], then it exerts an influence on the noisy signal input $\{r(t)\}$ in the way that it **convolves** the noisy signal input $\{r(t)\}$ to obtain the signal estimates $\{\hat{s}(t)\}$:

$$\hat{s}(t) = f(t) * r(t) \equiv \int_{0^-}^t f(\tau)r(t-\tau)\mathrm{d}\tau, \tag{C.18}$$

where the function $f(t)$ represents the linear time-invariant filter.

In real-time applications where systems actually evolve through time, it is obviously that they have a starting time point for operation (abstracted as time $t = 0$) and that they cannot generate future output or take future input in advance, this is why the convolution integral in (C.18) is done only on the time interval bounded between the starting time point 0 and current time t. Besides, for real-time operational efficiency, signal estimates are usually obtained in recursive way [4].

However, here the author does not constrain consideration into the context of real-time applications where signals are available sequentially through time. Instead, suppose all signals are available as if off-line processing of signals is performed in retrospect. This assumption is suitable to image filtering, because the entire image can be regarded to be captured instantly if the short exposure time is neglected. Then the convolution integral in (C.18) can be extended as

$$\hat{s}(t) = f(t) * r(t) \equiv \int\limits_{-\infty}^{+\infty} f(\tau) r(t-\tau) \mathrm{d}\tau. \tag{C.19}$$

So the problem of designing the filter becomes the problem of designing the filter function $f(t)$ such that appropriate signal estimates $\{\hat{s}(t)\}$ can be obtained via convolution described in (C.19).

C.3.2 Minimum Mean Square Error Filtering: Modelling as Functional Optimization

Let $\{t\}$ denote $(-\infty, +\infty)$ or the interval limit $[-t_M, t_M]$ as $t_M \to \infty$. The fitness of a filter function $f(t)$ may be evaluated in the light of the mean square error between the signal estimates $\{\hat{s}(t)\}$ generated by $f(t)$ and the true signal input $\{s(t)\}$:

$$\begin{aligned} MSE[f(t)] &= \frac{1}{|\{t\}|} \int\limits_{\{t\}} (\hat{s}(t) - s(t))^2 \mathrm{d}t \\ &= \frac{1}{|\{t\}|} \int\limits_{\{t\}} \left(\int\limits_{\{\tau\}} f(\tau) r(t-\tau) \mathrm{d}\tau - s(t) \right)^2 \mathrm{d}t. \end{aligned} \tag{C.20}$$

The smaller the mean square error $MSE[f(t)]$ is, the higher the fitness of $f(t)$ is. The larger the mean square error $MSE[f(t)]$ is, the lower the fitness of $f(t)$ is. Therefore, the optimal filter function $\hat{f}(t)$ is the one that achieves the *minimum mean square error*:

$$\hat{f}(t) = \mathrm{argmin}_{f(t)}\ MSE[f(t)] = \mathrm{argmin}_{f(t)}\ \frac{1}{|\{t\}|} \int\limits_{\{t\}} \left(\int\limits_{\{\tau\}} f(\tau) r(t-\tau) \mathrm{d}\tau - s(t) \right)^2 \mathrm{d}t, \tag{C.21}$$

which is a *functional optimization* problem. The optimal $\hat{f}(t)$ obtained by solving (C.21) is the **Wiener filter** [2].

C.3.3 Solution to the Functional Optimization

To analyse (C.21), resort to **calculus of variations**. Consider infinitesimal variation $\Delta f(t)$ on the optimal filter function $f(t)$ which is assumed well-formed:

$$\begin{aligned}
\Delta MSE[f] = \quad & MSE[f+\Delta f] - MSE[f] \\
= \quad & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f+\Delta f) * (s+n) - s]^2 \mathrm{d}t - \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [f * (s+n) - s]^2 \mathrm{d}t \\
= \quad & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [((f+\Delta f) * s - s) + (f+\Delta f) * n]^2 \mathrm{d}t - \\
& \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f * s - s) + f * n]^2 \mathrm{d}t \\
& (\text{Note that the noise n is assumed independent}) \\
= \quad & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} \{[((f+\Delta f) * s - s)^2 - (f * s - s)^2] \\
& + [((f+\Delta f) * n)^2 - (f * n)^2]\} \mathrm{d}t \\
= \quad & \tfrac{2}{|\{t\}|} \left\{ \int\limits_{\{t\}} [(f * s - s)(\Delta f * s) + (f * n)(\Delta f * n)] \mathrm{d}t + o(\|\Delta f\|^2) \right\}.
\end{aligned}$$

Since the second-order infinitesimal terms in $o(\|\Delta f\|^2)$ is positive definite (left to readers for verification), the sufficient and necessary optimality condition for the functional optimization problem described in (C.21) is that

$$\tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f * s - s)(\Delta f * s) + (f * n)(\Delta f * n)] \mathrm{d}t = 0 \tag{C.22}$$

always holds for arbitrary infinitesimal variation $\Delta f(t)$. Further obtain

$$\begin{aligned}
& \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f * s - s)(\Delta f * s) + (f * n)(\Delta f * n)] \mathrm{d}t \\
= \quad & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} \left[(f * s - s) \left(\int\limits_{\{\tau\}} s(t-\tau) \Delta f(\tau) \mathrm{d}\tau \right) \right. \\
& \left. + (f * n) \left(\int\limits_{\{\tau\}} n(t-\tau) \Delta f(\tau) \mathrm{d}\tau \right) \right] \mathrm{d}t \\
= \quad & \int\limits_{\{\tau\}} \left\{ \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f * s - s) s(t-\tau) + (f * n) n(t-\tau)] \mathrm{d}t \right\} \Delta f(\tau) \mathrm{d}\tau.
\end{aligned}$$

So the optimality condition that (C.22) always holds for arbitrary infinitesimal variation $\Delta f(t)$ is equivalent to the optimality condition that

$$\tfrac{1}{|\{t\}|} \int\limits_{\{t\}} [(f * s - s) s(t-\tau) + (f * n) n(t-\tau)] \mathrm{d}t = 0 \tag{C.23}$$

holds for arbitrary τ. In above derivation, f, s, and n actually denote $f(t)$, $s(t)$, and $n(t)$ with index t.

$$\begin{aligned} & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} (f*s)s(t-\tau)\mathrm{d}t \\ = \; & \tfrac{1}{|\{t\}|} \int\limits_{\{t\}} \int\limits_{\{a\}} f(a)s(t-a)s(t-\tau)\mathrm{d}a\mathrm{d}t \\ = \; & \tfrac{1}{|\{t\}|} \int\limits_{\{a\}} \left[\int\limits_{\{t\}} s(t-a)s(t-\tau)\mathrm{d}t \right] f(a)\mathrm{d}a \\ = \; & \int\limits_{\{a\}} \left[\tfrac{1}{|\{t\}|} \int\limits_{\{t\}} s(t+\tau-a)s(t)\mathrm{d}t \right] f(a)\mathrm{d}a \\ = \; & \int\limits_{\{a\}} A_s(\tau-a)f(a)\mathrm{d}a = A_s(\tau)*f(\tau). \end{aligned}$$

Similarly

$$\tfrac{1}{|\{t\}|} \int\limits_{\{t\}} (f*n)n(t-\tau)\mathrm{d}t = A_n(\tau)*f(\tau).$$

Besides,

$$\tfrac{1}{|\{t\}|} \int\limits_{\{t\}} s(t)s(t-\tau)\mathrm{d}t = A_s(\tau).$$

So the optimality condition (C.23) is further equivalent to

$$A_s(\tau)*f(\tau) + A_n(\tau)*f(\tau) = A_s(\tau), \quad \forall \tau \in \{t\}, \tag{C.24}$$

which gives the solution to (C.21) implicitly.

As $\{n(t), t \in \{t\}\}$ is commonly modelled as white noises, then substitute (C.16) into (C.24), replace the variable τ by the variable t, and obtain

$$A_s(t)*f(t) + n_0^2\delta(t)*f(t) = A_s(t)*f(t) + n_0^2 f(t) = A_s(t), \quad \forall t \in \{t\}. \tag{C.25}$$

C.4 Representative Filters

C.4.1 Frequency Domain Wiener Filter

Suppose the signal input $\{s(t), t \in \{t\}\}$ is constrained to an interval $\{t\} = [-T/2, T/2]$. The signal input $\{s(t), t \in \{t\}\}$ can be extended periodically throughout $(-\infty, +\infty)$. Take advantage of the *Fourier transform* and represent $f(t)$ and $s(t)$ on $\{t\}$ as

$$f(t) = \textstyle\sum_{k=-\infty}^{+\infty} f_k e_k(t), \tag{C.26}$$

$$s(t) = \textstyle\sum_{k=-\infty}^{+\infty} s_k e_k(t), \tag{C.27}$$

where $e_k(t) \equiv \mathrm{e}^{2k\pi\mathrm{i}t/T}$. Since $s(t)$ is a real-value function, $s_k = s^*_{-k}$ $(k = \pm 1, \cdots)$.[4] As

$$\tfrac{1}{T} \int\limits_{-T/2}^{T/2} e_{k_1}(a+t)e_{k_2}(a)\mathrm{d}a = \begin{cases} 0, & -k_2 \neq k_1 \\ e_k(t), & -k_2 = k_1 = k \end{cases}$$

one has

$$\begin{aligned} A_s(t) &= \frac{1}{T}\int\limits_{-T/2}^{T/2}\left[\sum_{k=-\infty}^{+\infty} s_k e_k(a+t)\sum_{k=-\infty}^{+\infty} s_k e_k(a)\right]\mathrm{d}a \\ &= \sum_{k=-\infty}^{+\infty} s_k s_{-k} e_k = \sum_{k=-\infty}^{+\infty} |s_k|^2 e_k. \end{aligned} \tag{C.28}$$

One further has

$$e_{k_1} * e_{k_2} = \int\limits_{-T/2}^{T/2} \mathrm{e}^{2k_1\pi\mathrm{i}(t-a)/T}\mathrm{e}^{2k_2\pi\mathrm{i}a/T}\mathrm{d}a = \begin{cases} 0, & k_2 \neq k_1 \\ Te_k(t), & k_2 = k_1 = k \end{cases}$$

So

$$A_s(t) * f(t) = \sum_{k=-\infty}^{+\infty} |s_k|^2 e_k * \sum_{k=-\infty}^{+\infty} f_k e_k = \sum_{k=-\infty}^{+\infty} T|s_k|^2 f_k e_k. \tag{C.29}$$

Substitute (C.26), (C.28) and (C.29) into (C.25):

$$\sum_{k=-\infty}^{+\infty}(T|s_k|^2 + n_0^2) f_k e_k = \sum_{k=-\infty}^{+\infty} |s_k|^2 e_k.$$

Compare corresponding terms in the series on both sides and obtain

$$f_k = \frac{|s_k|^2}{T|s_k|^2+n_0^2}, \quad k = 0, \pm 1, \pm 2, \cdots, \tag{C.30}$$

from which the Wiener filter function $f(t)$ can be recovered.

In practical applications, the true signal input $s(t)$ is unknown, so (C.30) is important mainly in theoretical sense. To put (C.30) into practice, one may compute the Fourier coefficients of the noisy signal input $r(t)$, namely $\{r_k\}$ such that

$$r(t) = \sum_{k=-\infty}^{+\infty} r_k e_k(t), \text{ where } r_k = \frac{1}{T}\int\limits_{-T/2}^{T/2} r(t) e_{-k}(t)\mathrm{d}t,$$

and then replace $\{s_k\}$ in (C.30) by $\{r_k\}$:

$$f_k \approx \frac{|r_k|^2}{T|r_k|^2+n_0^2}, \quad k = 0, \pm 1, \pm 2, \cdots. \tag{C.31}$$

Besides, one may apply the Wiener filter approximated by (C.31) to compute $\hat{s}(t)$ namely an estimate of the true signal input $s(t)$, and substitute $\hat{s}(t)$ into (C.30) to refine $\{f_k\}$ and $\{s_k\}$ iteratively. One may also refine estimates of the constant n_0^2 that represents the white noise energy over unit index length.

C.4.2 Gaussian Filter

(C.30) tells

$$f_k = \frac{|s_k|^2}{T|s_k|^2+n_0^2} = \frac{|s_{-k}|^2}{T|s_{-k}|^2+n_0^2} = f_{-k}. \tag{C.32}$$

So

$$f(t) = \sum_{k=-\infty}^{+\infty} f_k e_k(t) = f_0 + \sum_{k=1}^{+\infty} f_k(e_k(t) + e_{-k}(t)) = f_0 + 2\sum_{k=1}^{+\infty} f_k \cos\left(\frac{2k\pi}{T}t\right), \tag{C.33}$$

which implies that $f(t)$ is an *even* function. Besides, all $f_k = f_{-k} = |s_k|^2/(T|s_k|^2 + n_0^2) \geqslant 0$ ($k = 0, \pm 1, \pm 2, \cdots$).

In practical applications, filtering of the noisy signal input $r(t)$ at an index t is usually performed by convolution only over a local neighbourhood of t instead of over the entire index span $\{t\} = [-T/2, T/2]$. In other words, the span of the filter $f(t)$ is constrained to a local neighbourhood of 0, denoted $[-t_F, t_F]$, which is much smaller than the entire index span (namely $t_F \ll T$). In such local neighbourhood $[-t_F, t_F]$, the filter $f(t)$ described in (C.33) is approximated by a quadratic function

$$f(t) \approx f_0 + 2\sum_{k=1}^{+\infty} f_k \left[1 - \frac{1}{2}\left(\frac{2k\pi}{T}t\right)^2\right] \equiv a_f - b_f t^2 \tag{C.34}$$

with

$$a_f \equiv f_0 + 2\sum_{k=1}^{+\infty} f_k > 0, \quad b_f \equiv \sum_{k=1}^{+\infty} f_k \left(\frac{2k\pi}{T}\right)^2 > 0,$$

which has a reversed bowl shape.

In practice, such quadratic approximation of reversed bowl shape is normally replaced by a Gaussian approximation of bell shape:

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{t^2}{2\sigma^2}}, \tag{C.35}$$

which enjoys several merits: First, it also approximates well the quadratic shape of the Wiener filter (C.33) in the crucial neighbourhood $[-t_F, t_F]$ of 0. Second, it is bounded throughout $(-\infty, +\infty)$. Third, it is any-order differentiable everywhere and appears extremely smooth.

The filter described in (C.35) is called the **Gaussian filter** . Besides above merits, the Gaussian filter enjoys another merit for multiple-dimensional signal filtering such as image filtering: a multiple-dimensional (N-D) Gaussian filter can be decoupled into several (N) one-dimensional Gaussian filters, which facilitates multiple-dimensional signal filtering .[5]

To demonstrate the decoupling mechanism, take a two-dimensional Gaussian filter as example:

$$f(u, v) \propto \mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} = \mathrm{e}^{-\frac{u^2}{2\sigma^2}} \mathrm{e}^{-\frac{v^2}{2\sigma^2}} \equiv f_u(u) f_v(v), \tag{C.36}$$

where

$$f_u(u) \equiv \mathrm{e}^{-\frac{u^2}{2\sigma^2}}, \quad f_v(v) = \mathrm{e}^{-\frac{v^2}{2\sigma^2}}.$$

Then

$$\begin{aligned} f(u,v)*s(u,v) &= \int\limits_{\{\tau_1\}}\int\limits_{\{\tau_2\}} f_u(\tau_1)f_v(\tau_2)s(u-\tau_1,v-\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2 \\ &= \int\limits_{\{\tau_1\}} f_u(\tau_1)\left[\int\limits_{\{\tau_2\}} f_v(\tau_2)s(u-\tau_1,v-\tau_2)\mathrm{d}\tau_2\right]\mathrm{d}\tau_1 \\ &= \int\limits_{\{\tau_1\}} f_u(\tau_1)[f_v(v)*s(u-\tau_1,v)]\mathrm{d}\tau_1 = f_u(u)*[f_v(v)*s(u,v)], \end{aligned}$$

which implies that the two-dimensional Gaussian filter based convolution $f(u,v)*s(u,v)$ can be realized by two rounds of one-dimensional Gaussian filter based convolution, namely first convolving $s(u, v)$ by $f_v(v)$ along the dimension of v to obtain an intermediate result and afterwards convolving the intermediate result by $f_u(u)$ along the dimension of u.

Matlab code for two-dimensional Gaussian filtering is given as follows.

GaussianFilter2D.m

```
%% Two-dimensional Gaussian Filter
function imgD = GaussianFilter2D(img, stdF, wszF)
    % stdF: Filter standard deviation
    % wszF: Filter whole size (an odd number) i.e. wszF = 2*szF
    +1
    if (nargin<2)
        stdF = 1.5; wszF = 2*round(3*stdF)+1;
    elseif (nargin<3)
        wszF = 2*round(3*stdF)+1;
    end
    % Generate the one-dimensional Gaussian convolution template
    szF = (wszF-1)/2; t = -szF:szF; ct = exp(-t.^2/(2*stdF^2));
    ct = ct/sum(ct); [H, W] = size(img); imgI = double(img);
    imgD = zeros(H+2*szF,W+2*szF);imgD(szF+1:szF+H,szF+1:szF+W)=
    imgI;
    % First 1D Gaussian filter based convolution along rows
    for c = 1:W
        imgI(:,c) = imgD(1+szF:H+szF,c:c+2*szF)*ct';
    end
    imgD(szF+1:szF+H,szF+1:szF+W) = imgI; % Get the intermediate
     result
    % Second 1D Gaussian filter based convolution along columns
    for r = 1:H
        imgI(r,:) = ct*imgD(r:r+2*szF,1+szF:W+szF);
    end
    imgD = imgI;
end
```

The implementation demonstrated in the Matlab code **GaussianFilter2D.m** follows the definition of one-dimensional Gaussian filtering rather literally. This is to facilitate readers' understanding of the Gaussian filter. Besides, the implementation itself works not badly. On the other hand, it is worth noting that in works which involve multi-resolution or scale-space processing (see Appendix E), more efficient implementation of Gaussian filtering does exist in practice [5–7].

Sometime one may not need to perform two-dimensional Gaussian filtering over the entire image, but instead to perform Gaussian filtering over a set of points that are much lesser than the set of all pixels in the image. In other words, one may need to perform Gaussian filtering over only a set of points that are rather sparsely spread in the image.

Matlab code for such special purpose of two-dimensional Gaussian filtering is given as follows.

GaussianFilterPoints2D.m

```
%% Two-dimensional Gaussian Filter for a set of points
function ptF = GaussianFilterPoints2D(img, pt, stdF, wszF)
    % stdF: Filter standard deviation
    % wszF: Filter whole size (an odd number) i.e. wszF = 2*szF
    +1
    if (nargin<3)
        stdF = 1.5; wszF = 2*round(3*stdF)+1;
    elseif (nargin<4)
        wszF = 2*round(3*stdF)+1;
    end
    % Generate the one-dimensional Gaussian convolution template
    szF = (wszF-1)/2; t = -szF:szF; ct = exp(-t.^2/(2*stdF^2));
    ct = ct/sum(ct); ct = reshape(ct'*ct,1,[]);
    [H, W] = size(img); u = pt(1,:); v = pt(2,:);
    ptF = [pt; double(img(H*(u-1)+v))];
    % Filter the given set of points
    ii = find(u>szF & u<=W-szF & v>szF & v<=H-szF);
    iPt = H*(u(ii)-1)+v(ii); nPt = length(iPt);
    uOff = reshape(repmat(t,2*szF+1,1),1,[]);
    vOff = reshape(repmat(t',1,2*szF+1),1,[]);
    iOff = H*uOff+vOff; nOff = length(iOff);
    ptF(3,ii) = ct*double(img(repmat(iPt,nOff,1)+repmat(iOff',1,
    nPt)));
end
```

For the special two-dimensional Gaussian filtering over a sparse set of points, one can no longer take advantage of the fact that the two-dimensional Gaussian filter based convolution can be realized by two rounds of one-dimensional Gaussian filter based convolution. However, since only a sparse set of points are to be filtered, even a literal implementation of the special two-dimensional Gaussian filtering tends to still have a considerable gain in computational efficiency, compared with even well-designed implementation of normal two-dimensional Gaussian filtering over the entire image.

C.4.3 Median Filter

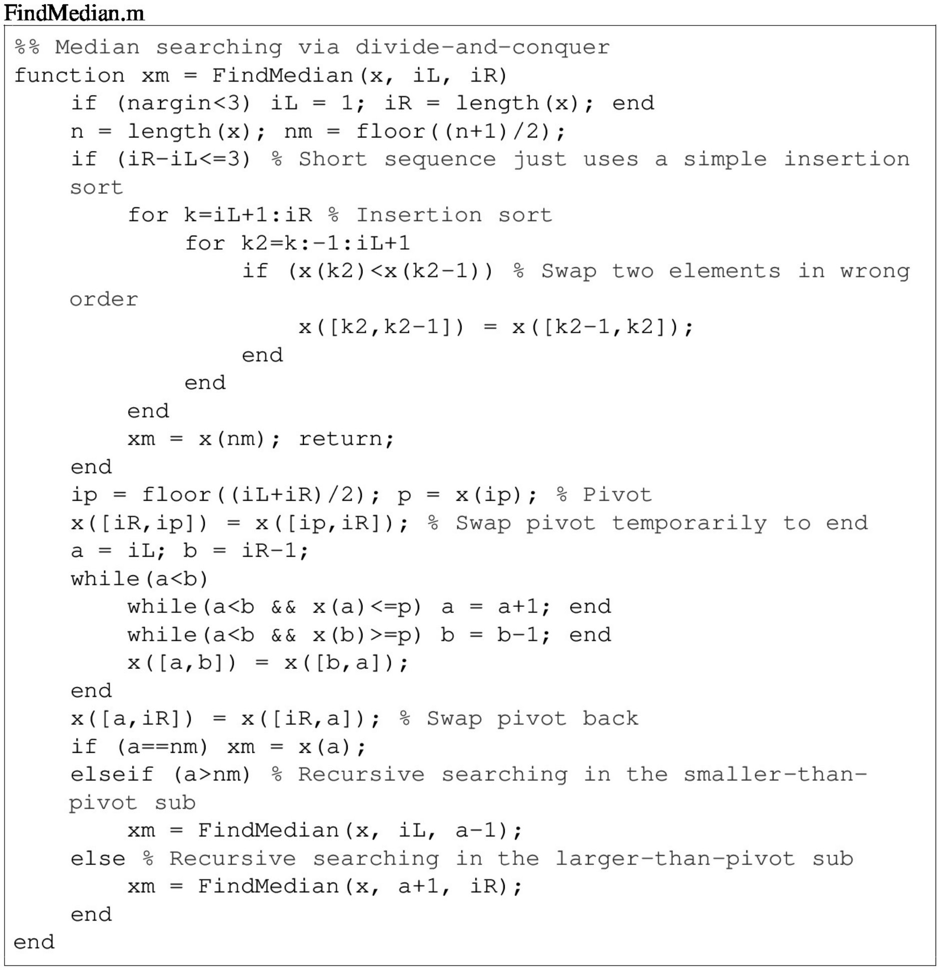
The **Median filter** [8] is a representative nonlinear filtering method that is intended for removing spike-like spurious signals while maintaining distinction among genuine signals as much as possible. The basic idea of the median filter is to replace the value of each pixel by the median of pixel values in a filter window centred at the pixel (i.e. pixel values in a neighbourhood of the pixel).

The efficiency of median filtering consists right in the efficiency of median searching in each filter window. Suppose the filter window size is n (an odd number) and hence the number of pixels in the filter window is $N = n^2$. It is time-consuming to perform complete-sorting based median searching namely to find the median pixel value of the filter window by first sorting all pixel values in the filter window and then selecting the middle one of the sorted pixel values. Complete-sorting based median searching for the filter window is of computational complexity $O(N \log N) = O(n^2 \log n^2) = O(n^2 \log n)$, if a standard sorting method such as the *Quicksort* [9] is used.

Such computational complexity can be reduced to $O(N) = O(n^2)$ immediately if the *divide-and-conquer* spirit similar to that of the Quicksort is instantiated for median searching. Given a sequence of elements, select a pivot (say the element at the middle position of the sequence), traverse the other elements, and assign each of them either to the smaller-than-pivot sub-sequence or to the larger-than-pivot sub-sequence. One can know the median must be located in which sub-sequence and continue median searching recursively inside this sub-sequence without considering the other sub-sequence any more. In such way of median searching, the computation complexity $C(N)$ follows roughly the recursive relationship

$$C(N) = C(N/2) + O(N)$$

which gives $C(N) = O(2N) = O(2n^2) = O(n^2)$. Matlab code for median searching that takes advantage of divide-and-conquer is given as follows.



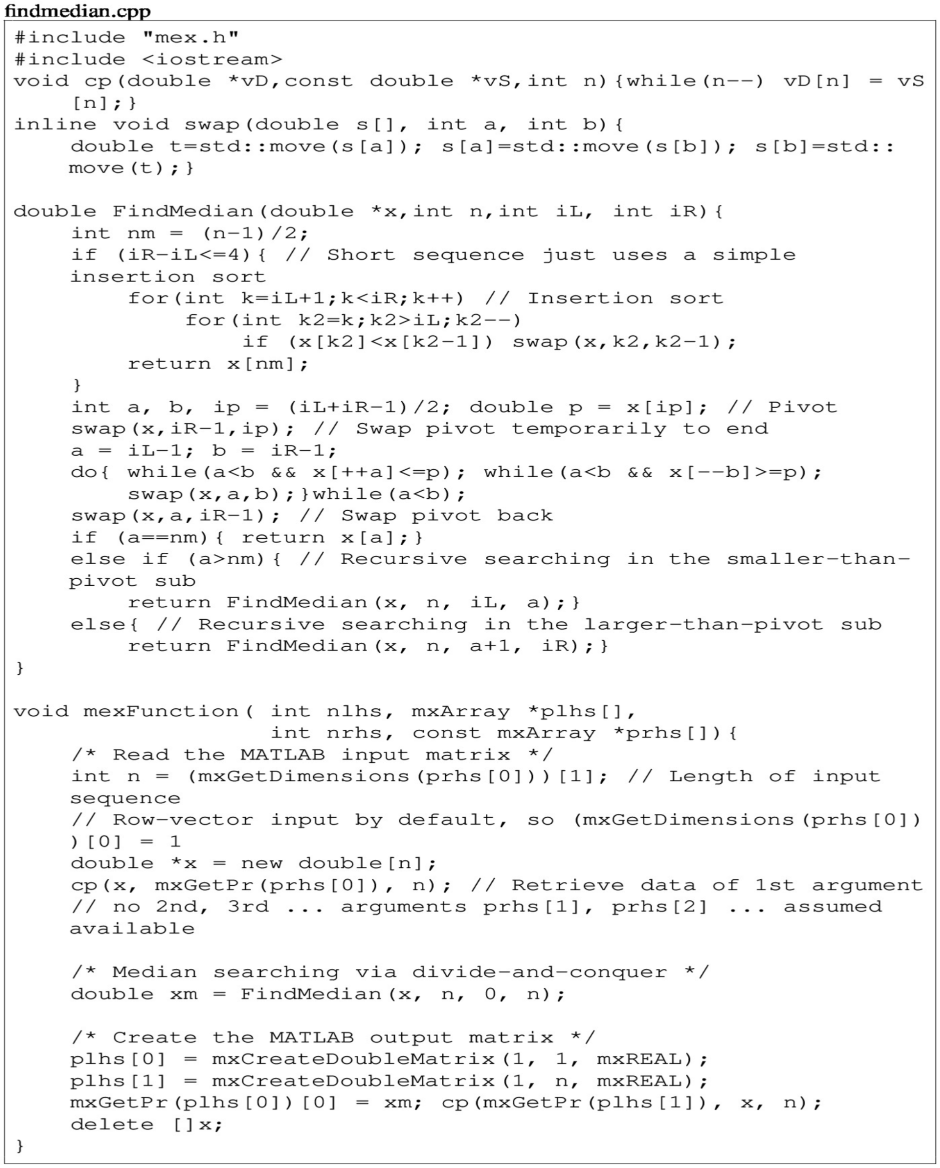
FindMedian.m

```
%% Median searching via divide-and-conquer
function xm = FindMedian(x, iL, iR)
    if (nargin<3) iL = 1; iR = length(x); end
    n = length(x); nm = floor((n+1)/2);
    if (iR-iL<=3) % Short sequence just uses a simple insertion
    sort
        for k=iL+1:iR % Insertion sort
            for k2=k:-1:iL+1
                if (x(k2)<x(k2-1)) % Swap two elements in wrong
    order
                    x([k2,k2-1]) = x([k2-1,k2]);
                end
            end
        end
        xm = x(nm); return;
    end
    ip = floor((iL+iR)/2); p = x(ip); % Pivot
    x([iR,ip]) = x([ip,iR]); % Swap pivot temporarily to end
    a = iL; b = iR-1;
    while(a<b)
        while(a<b && x(a)<=p) a = a+1; end
        while(a<b && x(b)>=p) b = b-1; end
        x([a,b]) = x([b,a]);
    end
    x([a,iR]) = x([iR,a]); % Swap pivot back
    if (a==nm) xm = x(a);
    elseif (a>nm) % Recursive searching in the smaller-than-
    pivot sub
        xm = FindMedian(x, iL, a-1);
    else % Recursive searching in the larger-than-pivot sub
        xm = FindMedian(x, a+1, iR);
    end
end
```

In above Matlab demonstration code, explicit m-script implementation of the *FindMedian* function is given, simply to facilitate readers' understanding of median searching in the spirit of *divide-and-conquer* as well as readers' direct use of the code in Matlab. However, execution of the m-script itself in Matlab is not efficient, even compared with complete-sorting based median searching that uses the Matlab built-in function *sort*.

In fact, above Matlab code does not have any algorithmic issue that causes the slowness of its execution in Matlab. The slowness is due to that the demonstrated m-script implementation incurs many “unfriendly” coding factors such as explicit loops for Matlab and many unnecessary copies of data. On the other hand, if readers implement the same *FindMedian* function in the C/C++ programming language and compile it for use in Matlab, readers will see that median searching in the spirit of *divide-and-*

conquer indeed outperforms complete-sorting based median searching considerably in terms of computational efficiency. Relevant C/C++ code for median searching is given as follows.



findmedian.cpp

```
#include "mex.h"
#include <iostream>
void cp(double *vD,const double *vS,int n){while(n--) vD[n] = vS
    [n];}
inline void swap(double s[], int a, int b){
    double t=std::move(s[a]); s[a]=std::move(s[b]); s[b]=std::
    move(t);}

double FindMedian(double *x,int n,int iL, int iR){
    int nm = (n-1)/2;
    if (iR-iL<=4){ // Short sequence just uses a simple
    insertion sort
        for(int k=iL+1;k<iR;k++) // Insertion sort
            for(int k2=k;k2>iL;k2--)
                if (x[k2]<x[k2-1]) swap(x,k2,k2-1);
        return x[nm];
    }
    int a, b, ip = (iL+iR-1)/2; double p = x[ip]; // Pivot
    swap(x,iR-1,ip); // Swap pivot temporarily to end
    a = iL-1; b = iR-1;
    do{ while(a<b && x[++a]<=p); while(a<b && x[--b]>=p);
        swap(x,a,b);}while(a<b);
    swap(x,a,iR-1); // Swap pivot back
    if (a==nm){ return x[a];}
    else if (a>nm){ // Recursive searching in the smaller-than-
    pivot sub
        return FindMedian(x, n, iL, a);}
    else{ // Recursive searching in the larger-than-pivot sub
        return FindMedian(x, n, a+1, iR);}
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[]){
    /* Read the MATLAB input matrix */
    int n = (mxGetDimensions(prhs[0]))[1]; // Length of input
    sequence
    // Row-vector input by default, so (mxGetDimensions(prhs[0])
    )[0] = 1
    double *x = new double[n];
    cp(x, mxGetPr(prhs[0]), n); // Retrieve data of 1st argument
    // no 2nd, 3rd ... arguments prhs[1], prhs[2] ... assumed
    available

    /* Median searching via divide-and-conquer */
    double xm = FindMedian(x, n, 0, n);

    /* Create the MATLAB output matrix */
    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
    plhs[1] = mxCreateDoubleMatrix(1, n, mxREAL);
    mxGetPr(plhs[0])[0] = xm; cp(mxGetPr(plhs[1]), x, n);
    delete []x;
}
```

Use the Matlab built-in *mex* mechanism to compile above C/C++ demonstration code and then invoke the compiled **findmedian** function just like a Matlab built-in function.[6] as follows.

```
mex findmedian.cpp
[xm, xo] = findmedian(x);
```

The m-script implementation **FindMedian.m** and its C/C++ counterpart **findmedian.cpp** aim at generic median searching, without taking advantage of any *a priori* data statistics. However, in the context of image processing, *a priori* data statistics are indeed available, which can be exploited to further enhance computational efficiency of median searching. For example, one at least knows that pixel values in the filter window are bounded between 0 and 255, so one may first establish a histogram for the pixel values, then locate the histogram bin containing the median, and finally find the median within the histogram bin that contains it.

Moreover, for median filtering on a two-dimensional image, as the filter window slides, neighbouring filter windows have most part in common, so the histogram of the previous filter window can be re-used to quickly form that of the next filter window, namely to update the histogram of the previous filter window by deleting the left-most column of the previous filter window and adding the right-most column of the next filter window. This further enhances computational efficiency of median filtering in the context of image processing [8].

In conventional median filtering , each of pixel values in the filter window is considered once. On the other hand, multiple copies of pixel values at special locations in the filter window may be incorporated into the data sequence for median searching, which is the basic idea of weighted median filtering [10]. Besides, instead of being fixed, the filter window size can also be varied adaptively according to on-line data statistics [11].

C.5 Frequency-Domain Analysis

As mentioned in Sect. C.2, the filter module may be treated as a *linear time-invariant* dynamic system, as formalized in (C.18):

$$\hat{s}(t) = f(t) * r(t) \equiv \int_{0^-}^{t} f(\tau) r(t-\tau) \mathrm{d}\tau,$$

where the function $f(t)$ represents the linear time-invariant filter.

Besides time-domain analysis of filtering like how to derive (C.24) and (C.25) for the *Wiener filter* (C.21), one can also perform **frequency-domain analysis** of filtering, based on theory of Laplace transform.

The **Laplace transform** of a generic time-domain function $f(t)$ that satisfies the condition

$$\int_{0^-}^{\infty} |f(t)| \mathrm{e}^{-\sigma t} \mathrm{d}t < \infty \quad (\text{for some positive real } \sigma)$$

is defined as

$$F(s) \equiv L\{f(t)\} = \int_{0^-}^{\infty} f(t) \mathrm{e}^{-st} \mathrm{d}t, \tag{C.37}$$

where s denotes the *Laplace variable* and is assumed to be located in the s-domain of absolute convergence by default. The **inverse Laplace transform** from the s-domain function $F(s)$ to the original time-domain function $f(t)$ is given as

$$f(t) \equiv L^{-1}\{F(s)\} = \frac{1}{2\pi \mathrm{i}} \int_{\sigma - \mathrm{i}\infty}^{\sigma + \mathrm{i}\infty} F(s) \mathrm{e}^{+st} \mathrm{d}s. \tag{C.38}$$

common Laplace transform pairs are summarized in Table C.1.

Table C.1 Common Laplace transform pairs

Time-domain function *f*(*t*)	***s*-domain function *F*(*s*)**

Time-domain function $f(t)$	**s-domain function $F(s)$**
$\delta(t)$	1
$u(t)$	$\frac{1}{s}$
t^n	$\frac{n!}{s^{n+1}}$
e^{-at}	$\frac{1}{s+a}$
$\sin(\omega t)$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
$e^{-at}\sin(\omega t)$	$\frac{\omega}{(s+a)^2+\omega^2}$
$e^{-at}\cos(\omega t)$	$\frac{s+a}{(s+a)^2+\omega^2}$
$e^{-at}f(t)$ **where** $L\{f(t)\}=F(s)$	$F(s+a)$
e.g. $L\{e^{-at}t^n\}$	$\frac{n!}{(s+a)^{n+1}}$
$f^{(k)}(t) \equiv \frac{d^k f(t)}{dt^k}$	$s^k L\{f(t)\} - s^{k-1}f(0^-) - \ldots - f^{(k-1)}(0^-)$
$\int_{-\infty}^{t} f(t)dt$	$\frac{1}{s}L\{f(t)\} + \frac{1}{s}\int_{-\infty}^{0^-} f(t)dt$
$f(t) * g(t) \equiv \int_{0^-}^{\infty} f(\tau)g(t-\tau)d\tau$	$L\{g(t)\}L\{f(t)\} \quad \text{or} \quad L\{f(t)\}L\{g(t)\}$

Perform the Laplace transform on both sides of (C.18) and obtain

$$\hat{S}(s) = F(s)R(s) \Longleftrightarrow \frac{\hat{S}(s)}{R(s)} = F(s), \tag{C.39}$$

where

$$R(s) \equiv L\{r(t)\}, \quad \hat{S}(s) \equiv L\{\hat{s}(t)\}, \quad F(s) \equiv L\{f(t)\}.$$

In (C.39), the s-domain function $F(s)$, which characterizes the inherent causal-relationship between the raw signal input $r(t)$ and the signal estimate $\hat{s}(t)$, is called the **transfer function** of the filter module $f(t)$.

The s-domain function $F(s)$ is normally in a form of *fractional polynomial*. Let it be denoted generically as

$$F(s) = \frac{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_0}{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_0} \tag{C.40}$$

and its *characteristic polynomial*, i.e. the denominator be factored as

$$a_n s^n + a_{n-1}s^{n-1} + \ldots + a_0 = a_n (s+\alpha_1)^{m_1}(s+\alpha_2)^{m_2}\ldots(s+\alpha_N)^{m_N},$$

where $-\alpha_1$, $-\alpha_2$, ..., $-\alpha_N$ denote roots of the characteristic polynomial and m_1,m_2, ..., m_N denote their numbers respectively. By default, the filter is assumed stable, which implies that the roots $-\alpha_1$, $-\alpha_2$, ..., $-\alpha_N$ are all with negative real part, namely Re($-\alpha_1$) < 0, Re($-\alpha_2$)< 0, ..., Re($-\alpha_N$)< 0.

Let the raw signal input $r(t) = A\ \sin\ (\omega t)$ be sinusoidal input and neglect the filter's initial conditions, then signal estimate is derived as

$$\hat{S}(s) = F(s)R(s) = \frac{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_0}{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_0} \cdot \frac{A\omega}{s^2+\omega^2}. \tag{C.41}$$

It can be decomposed into the sum of partial fractions [3] as [7]

$$\hat{S}(s) = A\left(\frac{c_1}{s+\omega\mathrm{i}} + \frac{c_2}{s-\omega\mathrm{i}} + \sum_{j=1}^{N}\sum_{k=1}^{m_j}\frac{c_{j,k}}{(s+\alpha_j)^k}\right), \tag{C.42}$$

where the generically-denoted coefficients c_1, c_2, and $c_{j,k}$ ($j \in \{1,\ldots,N\}$, $k \in \{1,\ldots, m_j\}$) are determined according to the transfer function $F(s)$. These coefficients are

$$c_1 = \lim_{s\to-\omega\mathrm{i}}\ (s+\omega\mathrm{i})F(s)\frac{\omega}{s^2+\omega^2} = \frac{\omega F(-\omega\mathrm{i})}{-2\omega\mathrm{i}} = \frac{\mathrm{i}}{2}[F_{\mathrm{E}}(\omega\mathrm{i}) - F_{\mathrm{O}}(\omega\mathrm{i})], \tag{C.43}$$

$$c_2 = \lim_{s\to\omega\mathrm{i}}\ (s-\omega\mathrm{i})F(s)\frac{\omega}{s^2+\omega^2} = \frac{\omega F(\omega\mathrm{i})}{2\omega\mathrm{i}} = -\frac{\mathrm{i}}{2}[F_{\mathrm{E}}(\omega\mathrm{i}) + F_{\mathrm{O}}(\omega\mathrm{i})], \tag{C.44}$$

and

$$c_{j,k} = \{\text{certain value}\},$$

where $F_{\mathrm{O}}(s)$ and $F_{\mathrm{E}}(s)$ with subscripts O and E denote the sub-polynomials of $F(s)$ with odd orders and even orders respectively—$F(s)$ may be viewed as infinite power series—Substitute (C.43) and (C.44) into (C.42) and obtain

$$\begin{aligned}\hat{S}(s) &= A\frac{F_{\mathrm{E}}(\omega\mathrm{i})\omega - F_{\mathrm{O}}(\omega\mathrm{i})\mathrm{i}s}{s^2+\omega^2} + A\sum_{j=1}^{N}\sum_{k=1}^{m_j}\frac{c_{j,k}}{(s+\alpha_j)^k}\\ &= A\frac{F_{\mathrm{R}}(\omega\mathrm{i})\omega + F_{\mathrm{I}}(\omega\mathrm{i})s}{s^2+\omega^2} + A\sum_{j=1}^{N}\sum_{k=1}^{m_j}\frac{c_{j,k}}{(s+\alpha_j)^k},\end{aligned} \tag{C.45}$$

where $F_{\mathrm{R}}(\omega\mathrm{i})$ and $F_{\mathrm{I}}(\omega\mathrm{i})$ denote the *real part* and the *imaginary part* of $F(\omega\mathrm{i})$ respectively, namely

$$F(\omega\mathrm{i}) = F_{\mathrm{R}}(\omega\mathrm{i}) + F_{\mathrm{I}}(\omega\mathrm{i})\mathrm{i}.$$

Since $F_{\mathrm{E}}(\omega\mathrm{i})$ contributes to the real part of $F(\omega\mathrm{i})$ whereas $F_{\mathrm{O}}(\omega\mathrm{i})$ contributes to the imaginary part of $F(\omega\mathrm{i})$, one has

$$\begin{aligned}F_{\mathrm{E}}(\omega\mathrm{i}) &= F_{\mathrm{R}}(\omega\mathrm{i}),\\ F_{\mathrm{O}}(\omega\mathrm{i}) &= F_{\mathrm{I}}(\omega\mathrm{i})\mathrm{i} \Longleftrightarrow F_{\mathrm{I}}(\omega\mathrm{i}) = -F_{\mathrm{O}}(\omega\mathrm{i})\mathrm{i},\end{aligned}$$

which are used in derivation of (C.45). The original time-domain function of $\hat{S}(s)$ is recovered as

$$\hat{s}(t) = A(F_{\mathrm{R}}(\omega\mathrm{i})\ \sin\ (\omega t) + F_{\mathrm{I}}(\omega\mathrm{i})\ \cos\ (\omega t)) + A\sum_{j=1}^{N}\sum_{k=1}^{m_j}c_{j,k}\mathrm{e}^{-\alpha_j t}\frac{t^{k-1}}{(k-1)!} \tag{C.46}$$

and its steady-state is

$$\lim_{t\to\infty}\ \hat{s}(t) \ = A(F_{\mathrm{R}}(\omega\mathrm{i})\ \sin\ (\omega t) + F_{\mathrm{I}}(\omega\mathrm{i})\ \cos\ (\omega t)) = A|F(\omega\mathrm{i})|\ \sin\ (\omega t + \angle F(\omega\mathrm{i})), \tag{C.47}$$

where $|F(\omega \mathrm{i})|$ and $\angle F(\omega \mathrm{i})$ are respectively the *magnitude* and *phase* (or *phase angle*) of the **frequency transfer function** $F(\omega \mathrm{i})$ [3].

(C.47) conveys that the filter transforms sinusoidal input asymptotically into steady-state sinusoidal output via the frequency transfer function $F(\omega \mathrm{i})$. The main effect of the filter consists in scaling the amplitude of the sinusoidal input by a *gain* that is the same to the magnitude of $F(\omega \mathrm{i})$.

A desirable filter should allow intended signals to pass while suppressing unneeded signals such as noises. In many practical applications, intended signals and unneeded signals have distinct frequency ranges on the whole, then the principle of designing a desirable filter is to let its frequency transfer function $F(\omega \mathrm{i})$ maintain a magnitude equal or close to 1 for ω in the frequency range associated with intended signals and maintain a magnitude equal or close to 0 for ω in the frequency range associated with unneeded signals.

For example, in practical applications, operational signals tend to be of low frequency, whereas noisy signals tend to be of high frequency. Then to have a filter that allow operational signals to pass while suppressing noisy signals, one may design a *low-pass filter* [13] such as that with the frequency transfer function

$$F(\omega \mathrm{i}) = \frac{1}{1 + \dfrac{\omega}{\omega_0}\mathrm{i}}$$

or equivalently with the transfer function

$$F(s) = \frac{1}{1 + \dfrac{s}{\omega_0}}.$$

Roughly speaking, such concrete low-pass filter allows signals of frequencies lower than ω_0 to pass while suppressing signals of frequencies larger than ω_0.

Footnotes

[1] According to the philosophy of “Yi Jing” or to the physical theory of quantum mechanics, nothing is absolutely determined. However, the author refrains from such philosophical or physical scrutiny, but uses the terms “deterministic” and “non-deterministic” just in a natural way consistent with humans’ common senses in daily life.

[2] A set is said to be countable if a bijective namely one-to-one correspondence can be established between its elements and numbers of the natural number set. In simple words, one can count all its elements one by one as one, two, three, and so on, even imaginarily to infinity. Finite sets are definitely countable sets, but not *vice versa*. The set of rational numbers is infinite but countable. The set of real numbers within any unbroken interval on the real-value axis is uncountable.

[3] Note that input signals are assumed to be generated according to a common probability distribution and the number of signals is usually large, then approximate computation of the expectation, variance, and auto-correlation in the spirit of the *strong law of large numbers*, as in (C.13).

[4] Consider the complex conjugate of the real-value function s, namely s^*, which is s itself. Then

$$\textstyle\sum_{k=-\infty}^{+\infty} s_k e_k = s = s^* = \sum_{k=-\infty}^{+\infty} s_k^* e_k^* = \sum_{k=-\infty}^{+\infty} s_k^* e_{-k},$$

which implies that $s_k = s_{-k}^*$ $(k = \pm 1, \pm 2, \cdots)$.

[5] Consider N-dimensional filtering with $N > 1$ and suppose the discrete filtering span for digital processing is M, then a generic N-dimensional filter involves convolution of computational complexity $O(M^N)$ at each signal point, whereas the N-dimensional Gaussian filter involves convolution only of computational complexity $O(NM)$ at each signal point.

[6] If a compatible C/C++ compiler has not been installed yet, then install one before compilation via *mex*.

[7] *Partial fraction decomposition* was introduced by *Euler* in the first volume (*tomus primus*) of his famous book *Introductio in analysin infinitorum* [12].

Appendix D

Curves and Surfaces

Some background knowledge on analysing curves and surfaces in the Euclidean space $\mathbb{R}^3$ via differential geometry [14, 15] is introduced.

D.1 Representation of Curves and Surfaces

Common ways to represent a two-dimensional curve (namely a curve on a plane) include:

- *Function representation*: Suppose a generic point is denoted (x, y) in the Cartesian coordinates system established on the plane, a two-dimensional curve may be represented by a function:

$$y = f(x) \text{ or } \mathrm{x} = \mathrm{f}^{-1}(\mathrm{y}). \tag{D.1}$$

- *Parametric representation*: A two-dimensional curve may be parametrized by a unique variable (say t), namely the x-part and the y-part of the curve are represented respectively by two functions in terms of the common parametric variable t:

$$\boldsymbol{p}(t) \equiv \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}. \tag{D.2}$$

- *Implicit representation*: A two-dimensional curve may also be represented implicitly by a bivariate equation in terms of x and y:

$$F(x, y) = 0. \tag{D.3}$$

Similarly, common ways to represent a three-dimensional curve (namely a curve in the Euclidean space $\mathbb{R}^3$) include:

- *Parametric representation*: A three-dimensional curve may be parametrized by a unique variable (say t), namely the x-part, the y-part, and the z-part of the curve are represented respectively by three functions in terms of the common parametric variable t:

$$\boldsymbol{p}(t) \equiv \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}. \tag{D.4}$$

- *Implicit representation*: A three-dimensional curve may also be represented implicitly by a group of trivariate equations in terms of x, y, and z:

$$\begin{cases} F_1(x, y, z) = 0 \\ F_2(x, y, z) = 0 \end{cases} \tag{D.5}$$

Common ways to represent a surface (by default a three-dimensional surface) in the Euclidean space $\mathbb{R}^3$ include:

- *Parametric representation*: A surface may be parametrized by a pair of variables (say (u, v)), namely the x-part, the y-part, and the z-part of the surface are represented respectively by three bivariate functions in terms of the common pair of parametric variables (u, v):

$$\boldsymbol{p}(u, v) \equiv \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}. \tag{D.6}$$

- *Implicit representation*: A surface may also be represented implicitly by a trivariate equation in terms of x, y, and z:

$$F(x, y, z) = 0. \tag{D.7}$$

D.2 Curvatures of Curves and Surfaces

D.2.1 Frenet Formula and Curvatures of Curves

Derivation of curvature formulas for curves resorts to the first equality (D.8a) of the **Serret-Frenet formula** also known as the **Frenet formula** [15, 16]

$$\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = k\boldsymbol{N}, \tag{D.8a}$$

$$\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -k\boldsymbol{T} + \tau\boldsymbol{B}, \tag{D.8b}$$

$$\frac{\mathrm{d}\boldsymbol{B}}{\mathrm{d}s} = -\tau\boldsymbol{N}, \tag{D.8c}$$

where k denotes the *curvature*, $\boldsymbol{T}$ denotes the unit *tangent vector*, $\boldsymbol{N}$ denotes the unit *normal vector*, τ denotes the *torsion*, and $\boldsymbol{B} \equiv \boldsymbol{T} \times \boldsymbol{N}$ denotes the unit *binormal vector*.[8] (D.8a) leads to the Frenet unsigned curvature formula

$$|k| = \left|\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s}\right| \tag{D.9}$$

and the Frenet signed curvature formula

$$k = \boldsymbol{N} \cdot k\boldsymbol{N} = \boldsymbol{N} \cdot \frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = \frac{\mathrm{d}}{\mathrm{d}s}(\boldsymbol{N} \cdot \boldsymbol{T}) - \frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T} = -\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T}. \tag{D.10}$$

If the curvature sign is neglected, (D.10) becomes

$$|k| = \left|\boldsymbol{N} \cdot \frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s}\right| = \left|\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T}\right|.$$

Curvature formulas for a planar curve are derived according to its different representations respectively:

- *Function representation based curvature formula*: Given a generic point on a planar curve represented by the function $y = f(x)$ in (D.1), its unit tangent vector is

  $$\boldsymbol{T} = \frac{1}{\sqrt{\mathrm{d}x^2+\mathrm{d}y^2}}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}y\end{bmatrix} = \frac{1}{\sqrt{1+y'^2}}\begin{bmatrix}1\\ y'\end{bmatrix}$$

  and its unit normal vector is

  $$\boldsymbol{N} = \frac{1}{\sqrt{\mathrm{d}x^2+\mathrm{d}y^2}}\begin{bmatrix}-\mathrm{d}y\\ \mathrm{d}x\end{bmatrix} = \frac{1}{\sqrt{1+y'^2}}\begin{bmatrix}-y'\\ 1\end{bmatrix}.$$

  Its curvature is

  $$k = -\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T} = \frac{\frac{y''}{(1+y'^2)^{\frac{3}{2}}}\begin{bmatrix}1\\ y'\end{bmatrix}\mathrm{d}x}{\sqrt{1+y'^2}\mathrm{d}x} \cdot \frac{1}{\sqrt{1+y'^2}}\begin{bmatrix}1\\ y'\end{bmatrix} = \frac{y''}{(1+y'^2)^{\frac{3}{2}}}, \tag{D.11}$$

  where $y' \equiv \frac{\mathrm{d}y}{\mathrm{d}x}$ and $y'' \equiv \frac{\mathrm{d}^2y}{\mathrm{d}x^2}$.

- *Parametric representation based curvature formula*: Given a generic point on a planar curve represented parametrically by (D.2), its unit tangent vector is

  $$\boldsymbol{T} = \frac{1}{\sqrt{x'^2+y'^2}}\begin{bmatrix}x'\\ y'\end{bmatrix}$$

  and its unit normal vector is

  $$\boldsymbol{N} = \frac{1}{\sqrt{x'^2+y'^2}}\begin{bmatrix}-y'\\ x'\end{bmatrix}.$$

Its curvature is

$$\begin{aligned} k &= -\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T} = -\frac{\frac{1}{(x'^2+y'^2)^{\frac{3}{2}}} \begin{bmatrix} -y''x'^2 + x''x'y' \\ x''y'^2 - y''x'y' \end{bmatrix} \mathrm{d}t}{\sqrt{x'^2+y'^2}\,\mathrm{d}t} \cdot \frac{1}{\sqrt{x'^2+y'^2}} \begin{bmatrix} x' \\ y' \end{bmatrix} \\ &= \frac{y''x' - x''y'}{(x'^2+y'^2)^{\frac{3}{2}}}, \end{aligned} \tag{D.12}$$

where $x' \equiv \frac{\mathrm{d}x}{\mathrm{d}t}$, $y' \equiv \frac{\mathrm{d}y}{\mathrm{d}t}$, $x'' \equiv \frac{\mathrm{d}^2x}{\mathrm{d}t}$, and $y'' \equiv \frac{\mathrm{d}^2y}{\mathrm{d}t}$.

- *Implicit representation based curvature formula*: Given a generic point on a planar curve represented implicitly by (D.3), its unit normal vector is the normalized gradient of $F(x, y)$, namely

$$\boldsymbol{N} = \frac{\nabla F}{|\nabla F|} = \frac{1}{\sqrt{F_x^2+F_y^2}} \begin{bmatrix} F_x \\ F_y \end{bmatrix},$$

and its unit tangent vector is

$$\boldsymbol{T} = \frac{1}{\sqrt{F_x^2+F_y^2}} \begin{bmatrix} -F_y \\ F_x \end{bmatrix}.$$

Its curvature is

$$\begin{aligned} k &= -\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \cdot \boldsymbol{T} = -\boldsymbol{T}^{\mathrm{T}} \begin{bmatrix} \frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}x} & \frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}y} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}x}{\mathrm{d}s} \\ {[9pt]}\,\frac{\mathrm{d}y}{\mathrm{d}s} \end{bmatrix} \\ &= -\boldsymbol{T}^{\mathrm{T}} \nabla(\boldsymbol{N}) \boldsymbol{T} = -\boldsymbol{T}^{\mathrm{T}} \nabla \left(\frac{\nabla F}{|\nabla F|} \right) \boldsymbol{T} \\ &= -\boldsymbol{T}^{\mathrm{T}} \left(\frac{\nabla^2 F |\nabla F| - \nabla F \nabla |\nabla F|}{|\nabla F|^2} \right) \boldsymbol{T} = -\boldsymbol{T}^{\mathrm{T}} \frac{\nabla^2 F}{|\nabla F|} \boldsymbol{T} \quad (\text{note } \ \boldsymbol{T}^{\mathrm{T}} \nabla F = 0) \\ &= -\frac{1}{(F_x^2+F_y^2)^{\frac{3}{2}}} \begin{bmatrix} -F_y & F_x \end{bmatrix} \begin{bmatrix} F_{xx} & F_{xy} \\ F_{xy} & F_{yy} \end{bmatrix} \begin{bmatrix} -F_y \\ F_x \end{bmatrix}. \end{aligned} \tag{D.13}$$

In fact, the way of derivation in (D.13) can also be applied to derive curvature formulas for a function curve and a parametric curve . For a function curve described by (D.1), rewrite it as an implicit form

$$F(x, y) = f(x) - y = 0$$

and apply (D.13) directly to obtain

$$k=-\frac{1}{(y'^2+(-1)^2)^{\frac{3}{2}}}\begin{bmatrix}-(-1) & y'\end{bmatrix}\begin{bmatrix}y'' & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}-(-1)\\ y'\end{bmatrix}=\frac{y''}{(1+y'^2)^{\frac{3}{2}}},$$

which is the same to (D.11). For a parametric curve described by (D.2), to follow the literal derivation formalism in (D.13), one may express t "virtually" as a bivariate function explicitly in terms of x and y:

$$t=\alpha x^{-1}(x)+(1-\alpha)y^{-1}(y),$$

where $x^{-1}()$ and $y^{-1}()$ denote the inverse functions of $x(t)$ and $y(t)$ respectively. Replace the gradient ∇F in (D.13) by the normal vector $\boldsymbol{G}=\begin{bmatrix}-y' & x'\end{bmatrix}^{\mathrm{T}}$ and obtain

$$\begin{aligned}
k = \quad & -\boldsymbol{T}^{\mathrm{T}}\nabla\left(\frac{\boldsymbol{G}}{|\boldsymbol{G}|}\right)\boldsymbol{T}=-\boldsymbol{T}^{\mathrm{T}}\left(\frac{\nabla\boldsymbol{G}|\boldsymbol{G}|-\boldsymbol{G}\nabla|\boldsymbol{G}|}{|\boldsymbol{G}|^2}\right)\boldsymbol{T}=-\boldsymbol{T}^{\mathrm{T}}\frac{\nabla\boldsymbol{G}}{|\boldsymbol{G}|}\boldsymbol{T}\\
= \quad & -\boldsymbol{T}^{\mathrm{T}}\frac{1}{|\boldsymbol{G}|}\begin{bmatrix}-\frac{\partial y'}{\partial t}\frac{\partial t}{\partial x} & -\frac{\partial y'}{\partial t}\frac{\partial t}{\partial y}\\ {[6pt]}\frac{\partial x'}{\partial t}\frac{\partial t}{\partial x} & \frac{\partial x'}{\partial t}\frac{\partial t}{\partial y}\end{bmatrix}\boldsymbol{T}\\
= \quad & -\boldsymbol{T}^{\mathrm{T}}\frac{1}{|\boldsymbol{G}|}\begin{bmatrix}-y''\alpha\frac{\partial x^{-1}(x)}{\partial x} & -y''(1-\alpha)\frac{\partial y^{-1}(y)}{\partial y}\\ {[6pt]}x''\alpha\frac{\partial x^{-1}(x)}{\partial x} & x''(1-\alpha)\frac{\partial y^{-1}(y)}{\partial y}\end{bmatrix}\boldsymbol{T}\\
= \quad & -\frac{1}{\sqrt{x'^2+y'^2}}\begin{bmatrix}x' & y'\end{bmatrix}\frac{1}{\sqrt{x'^2+y'^2}}\\
& \begin{bmatrix}-y''\alpha\frac{1}{x'} & -y''(1-\alpha)\frac{1}{y'}\\ {[6pt]}x''\alpha\frac{1}{x'} & x''(1-\alpha)\frac{1}{y'}\end{bmatrix}\frac{1}{\sqrt{x'^2+y'^2}}\begin{bmatrix}x'\\ y'\end{bmatrix}\\
= \quad & -\frac{1}{(x'^2+y'^2)^{\frac{3}{2}}}\begin{bmatrix}x' & y'\end{bmatrix}\begin{bmatrix}-y''\\ x''\end{bmatrix}\\
= \quad & \frac{y''x'-x''y'}{(x'^2+y'^2)^{\frac{3}{2}}}\quad \text{(independent of the constant } \alpha\text{)},
\end{aligned}$$

which is the same to (D.12). Instead of following the literal derivation formalism in (D.13), one can follow spirit of derivation in (D.13):

$$\begin{aligned}
k &= -\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s}\cdot\boldsymbol{T}=-\boldsymbol{T}^{\mathrm{T}}\frac{\mathrm{d}\boldsymbol{N}}{\sqrt{x'^2+y'^2}\mathrm{d}t}=-\frac{1}{\sqrt{x'^2+y'^2}}\boldsymbol{T}^{\mathrm{T}}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\boldsymbol{G}}{|\boldsymbol{G}|}\right)\\
&= -\frac{1}{\sqrt{x'^2+y'^2}}\boldsymbol{T}^{\mathrm{T}}\left(\frac{\boldsymbol{G}'|\boldsymbol{G}|-\boldsymbol{G}|\boldsymbol{G}|'}{|\boldsymbol{G}|^2}\right)=-\frac{1}{\sqrt{x'^2+y'^2}}\frac{\boldsymbol{T}^{\mathrm{T}}\boldsymbol{G}'}{|\boldsymbol{G}|}\\
&= -\frac{1}{(x'^2+y'^2)^{\frac{3}{2}}}\begin{bmatrix}x' & y'\end{bmatrix}\begin{bmatrix}-y''\\ x''\end{bmatrix}=\frac{y''x'-x''y'}{(x'^2+y'^2)^{\frac{3}{2}}},
\end{aligned}$$

which is apparently more convenient than previous derivation.

Curvature formulas for a three-dimensional curve can also be derived according to its concrete representations.

- *Parametric representation based curvature formula*: Given a generic point on a three-dimensional curve represented parametrically by (D.4), its tangent vector is

$$\boldsymbol{p}' \equiv \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

  and its unit tangent vector is $\boldsymbol{T} = \boldsymbol{p}'/|\boldsymbol{p}'|$ (note $|\boldsymbol{p}'| = \sqrt{\boldsymbol{p}'^{\mathrm{T}}\boldsymbol{p}'}$). The Frenet formula gives

$$\begin{aligned} k\boldsymbol{N} &= \frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = \frac{\mathrm{d}}{|\boldsymbol{p}'|\mathrm{d}t}(\boldsymbol{p}'/|\boldsymbol{p}'|) = \frac{1}{|\boldsymbol{p}'|^3}(\boldsymbol{p}''|\boldsymbol{p}'|-\boldsymbol{p}'|\boldsymbol{p}'|') \\ &= \frac{1}{|\boldsymbol{p}'|^3}\frac{|\boldsymbol{p}'|^2\boldsymbol{I}-\boldsymbol{p}'\boldsymbol{p}'^{\mathrm{T}}}{|\boldsymbol{p}'|}\boldsymbol{p}'' = \frac{1}{|\boldsymbol{p}'|^3}\frac{\boldsymbol{S}_{\boldsymbol{p}'}^{\mathrm{T}}\boldsymbol{S}_{\boldsymbol{p}'}}{|\boldsymbol{p}'|}\boldsymbol{p}'' = -\frac{1}{|\boldsymbol{p}'|^3}\frac{\boldsymbol{S}_{\boldsymbol{p}'}^2}{|\boldsymbol{p}'|}\boldsymbol{p}'' \\ &= -\frac{1}{|\boldsymbol{p}'|^3}\frac{\boldsymbol{p}'\times(\boldsymbol{p}'\times\boldsymbol{p}'')}{|\boldsymbol{p}'|} \end{aligned} \tag{D.14}$$

  and the unsigned curvature is

$$|k| = \frac{|\boldsymbol{p}'\times\boldsymbol{p}''|}{|\boldsymbol{p}'|^3} = \frac{\sqrt{(y''x'-x''y')^2+(z''y'-y''z')^2+(x''z'-z''x')^2}}{(x'^2+y'^2+z'^2)^{\frac{3}{2}}}, \tag{D.15}$$

  where $\boldsymbol{S}_{\boldsymbol{p}'}$ denotes the skew-symmetric matrix of $\boldsymbol{p}'$.[9] Apparently, (D.12) is the two-dimensional special case of (D.15) where $z \equiv 0$.

- *Implicit representation based curvature formula*: Given a generic point on a planar curve represented implicitly by (D.5), its tangent vector is

$$\boldsymbol{R} = \nabla F_1 \times \nabla F_2$$

  and its unit tangent vector is $\boldsymbol{T} = \boldsymbol{R}/|\boldsymbol{R}|$. The Frenet formula gives

$$\begin{aligned} k\boldsymbol{N} &= \frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = \begin{bmatrix} \frac{1}{\mathrm{d}x} & \frac{1}{\mathrm{d}y} & \frac{1}{\mathrm{d}z} \end{bmatrix} \boldsymbol{T} \begin{bmatrix} \mathrm{d}x \\ \mathrm{d}y \\ \mathrm{d}z \end{bmatrix} \frac{1}{\mathrm{d}s} = \nabla(\boldsymbol{T})\boldsymbol{T} = \nabla\left(\frac{\boldsymbol{R}}{|\boldsymbol{R}|}\right)\frac{\boldsymbol{R}}{|\boldsymbol{R}|} \\ &= \left(\frac{\nabla\boldsymbol{R}|\boldsymbol{R}| - \boldsymbol{R}\nabla|\boldsymbol{R}|}{|\boldsymbol{R}|^2}\right)\frac{\boldsymbol{R}}{|\boldsymbol{R}|} = \frac{1}{|\boldsymbol{R}|^3}\frac{(|\boldsymbol{R}|^2\boldsymbol{I} - \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}})\nabla\boldsymbol{R}}{|\boldsymbol{R}|}\boldsymbol{R} \\ &= \frac{1}{|\boldsymbol{R}|^3}\frac{\boldsymbol{S}_{\boldsymbol{R}}^{\mathrm{T}}\boldsymbol{S}_{\boldsymbol{R}}\nabla\boldsymbol{R}}{|\boldsymbol{R}|}\boldsymbol{R} = -\frac{1}{|\boldsymbol{R}|^3}\frac{\boldsymbol{S}_{\boldsymbol{R}}^2\nabla\boldsymbol{R}}{|\boldsymbol{R}|}\boldsymbol{R} \\ &= -\frac{1}{|\boldsymbol{R}|^3}\frac{\boldsymbol{R}\times(\boldsymbol{R}\times(\nabla\boldsymbol{R})\boldsymbol{R})}{|\boldsymbol{R}|} \end{aligned} \tag{D.16}$$

and the unsigned curvature is

$$|k| = \frac{|\boldsymbol{R}\times(\nabla\boldsymbol{R})\boldsymbol{R}|}{|\boldsymbol{R}|^3} = \frac{|(\nabla F_1\times\nabla F_2)\times((\nabla(\nabla F_1\times\nabla F_2))(\nabla F_1\times\nabla F_2))|}{|\nabla F_1\times\nabla F_2|^3}, \tag{D.17}$$

where $\nabla\boldsymbol{R} = \nabla(\nabla F_1 \times \nabla F_2)$ may be computed according to firstly-computed $\boldsymbol{R} = \nabla F_1 \times \nabla F_2$ or computed as

$$\nabla(\nabla F_1 \times \nabla F_2) = \boldsymbol{S}_{\nabla F_1}\nabla^2 F_2 - \boldsymbol{S}_{\nabla F_2}\nabla^2 F_1.$$

For example, consider an implicit planar curve $F(x, y) = 0$ which can also be treated as a three-dimensional curve represented implicitly by $F_1(x, y, z) = F(x, y) = 0$ and $F_2(x, y, z) = z = 0$. So

$$\begin{aligned} \nabla F_1 &= [F_x \quad F_y \quad 0]^{\mathrm{T}}, \quad \nabla F_2 = [0 \quad 0 \quad 1]^{\mathrm{T}}, \\ \boldsymbol{R} &= \nabla F_1 \times \nabla F_2 = [F_y \quad -F_x \quad 0]^{\mathrm{T}}, \\ \nabla\boldsymbol{R} &= \begin{bmatrix} F_{xy} & F_{yy} & 0 \\ -F_{xx} & -F_{xy} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (\nabla\boldsymbol{R})\boldsymbol{R} = \begin{bmatrix} F_{xy}F_y - F_{yy}F_x \\ -F_{xx}F_y + F_{xy}F_x \\ 0 \end{bmatrix}, \\ \boldsymbol{R}\times(\nabla\boldsymbol{R})\boldsymbol{R} &= \begin{bmatrix} 0 \\ 0 \\ -F_{xx}F_y^2 + 2F_{xy}F_xF_y - F_{yy}F_x^2 \end{bmatrix}, \\ |k| &= \frac{|F_{xx}F_y^2 - 2F_{xy}F_xF_y + F_{yy}F_x^2|}{(F_x^2+F_y^2)^{\frac{3}{2}}} \\ &= \frac{1}{(F_x^2+F_y^2)^{\frac{3}{2}}}\left|\begin{bmatrix} -F_y & F_x \end{bmatrix}\begin{bmatrix} F_{xx} & F_{xy} \\ F_{xy} & F_{yy} \end{bmatrix}\begin{bmatrix} -F_y \\ F_x \end{bmatrix}\right|, \end{aligned}$$

which is consistent with (D.13).

D.2.2 Curvature Formulas for Parametric Surfaces

Given a generic point on a surface represented parametrically by (D.6), its tangent plane is the plane $< \boldsymbol{p}_u, \boldsymbol{p}_v >$[10] spanned by the tangent vector $\boldsymbol{p}_u \equiv \frac{\partial \boldsymbol{p}}{\partial u}$ and the tangent vector $\boldsymbol{p}_v \equiv \frac{\partial \boldsymbol{p}}{\partial v}$, and its normal vector is $\boldsymbol{G} = \boldsymbol{p}_u \times \boldsymbol{p}_v$. Each tangent vector on the tangent plane is a linear combination of the tangent vector $\boldsymbol{p}_u$ and the tangent vector $\boldsymbol{p}_v$. Denote

$$\boldsymbol{R}_{a,b}\ (\text{or simply}\,\mathbf{R}) \equiv \mathrm{a}\mathbf{p}_\mathrm{u} + \mathrm{b}\mathbf{p}_\mathrm{v}$$

and consider the curvature $k_{a,b}$ associated with the tangent vector $\boldsymbol{R}_{a,b}$, namely the curvature of the intersected curve between the plane $< \boldsymbol{R}_{a,b}, \boldsymbol{G} >$ spanned by the tangent vector $\boldsymbol{R}_{a,b}$ and the normal vector $\boldsymbol{G}$ and the given surface.

The surface differential $\mathrm{d}\boldsymbol{p} = \boldsymbol{p}_u \mathrm{d}u + \boldsymbol{p}_v \mathrm{d}v$ located on the intersected curve satisfies the constraint $(\boldsymbol{G} \times \boldsymbol{R}_{a,b}) \cdot d\boldsymbol{p} = 0$, so

$$\begin{aligned}
(\boldsymbol{G} \times \boldsymbol{R}_{a,b}) \cdot \mathrm{d}\boldsymbol{p} &= (\boldsymbol{G} \times (a\boldsymbol{p}_u + b\boldsymbol{p}_v)) \cdot (\boldsymbol{p}_u \mathrm{d}u + \boldsymbol{p}_v \mathrm{d}v) \\
&= a((\boldsymbol{G} \times \boldsymbol{p}_u) \cdot \boldsymbol{p}_v)\mathrm{d}v + b((\boldsymbol{G} \times \boldsymbol{p}_v) \cdot \boldsymbol{p}_u)\mathrm{d}u \\
&= ((\boldsymbol{p}_u \times \boldsymbol{p}_v) \cdot \boldsymbol{G})(a \cdot \mathrm{d}v - b \cdot \mathrm{d}u) = 0 \\
&\Rightarrow a \cdot \mathrm{d}v = b \cdot \mathrm{d}u \Longleftrightarrow \mathrm{d}u : \mathrm{d}v = \mathrm{a} : \mathrm{b},
\end{aligned}$$

the derivation of which involves the following equality

$$\begin{aligned}
(\boldsymbol{x} \times \boldsymbol{y}) \cdot \boldsymbol{z} &= -(\boldsymbol{y} \times \boldsymbol{x}) \cdot \boldsymbol{z} = -\boldsymbol{z}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{y}} \boldsymbol{x} = -\left(\boldsymbol{z}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{y}} \boldsymbol{x}\right)^{\mathrm{T}} = -\boldsymbol{x}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{y}}^{\mathrm{T}} \boldsymbol{z} \\
&= \boldsymbol{x}^{\mathrm{T}} \boldsymbol{S}_{\boldsymbol{y}} \boldsymbol{z} = (\boldsymbol{y} \times \boldsymbol{z}) \cdot \boldsymbol{x} = -(\boldsymbol{z} \times \boldsymbol{y}) \cdot \boldsymbol{x} \\
&\qquad (\text{proved similarly as above}) \\
&= (\boldsymbol{z} \times \boldsymbol{x}) \cdot \boldsymbol{y} = -(\boldsymbol{x} \times \boldsymbol{z}) \cdot \boldsymbol{y} \quad (\forall\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^3).
\end{aligned}$$

Besides,

$$\frac{\mathrm{d}u}{\mathrm{d}s} = \frac{\mathrm{d}u}{|\mathrm{d}\boldsymbol{p}|} = \frac{a}{|\boldsymbol{R}_{a,b}|}, \quad \frac{\mathrm{d}v}{\mathrm{d}s} = \frac{\mathrm{d}v}{|\mathrm{d}\boldsymbol{p}|} = \frac{b}{|\boldsymbol{R}_{a,b}|},$$

$$|\boldsymbol{R}_{a,b}|^2 = [a \quad b] \begin{bmatrix} \boldsymbol{p}_u \cdot \boldsymbol{p}_u & \boldsymbol{p}_u \cdot \boldsymbol{p}_v \\ \boldsymbol{p}_u \cdot \boldsymbol{p}_v & \boldsymbol{p}_v \cdot \boldsymbol{p}_v \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = [a \quad b] \begin{bmatrix} \boldsymbol{p}_u^{\mathrm{T}} \\ \boldsymbol{p}_v^{\mathrm{T}} \end{bmatrix} [\boldsymbol{p}_u \quad \boldsymbol{p}_v] \begin{bmatrix} a \\ b \end{bmatrix},$$

where the matrix $\boldsymbol{J}_{\mathrm{I}} \equiv [\boldsymbol{p}_u \quad \boldsymbol{p}_v]^{\mathrm{T}} [\boldsymbol{p}_u \quad \boldsymbol{p}_v]$ is called the *first fundamental form*.

The unit tangent vector along $\boldsymbol{R}_{a,b}$ is $\boldsymbol{T} = \boldsymbol{R}_{a,b} / |\boldsymbol{R}_{a,b}|$ and the unit normal vector is $\boldsymbol{N} = \boldsymbol{G} / |\boldsymbol{G}|$. The Frenet formula gives

$$
\begin{aligned}
k_{a,b} &= -\boldsymbol{T}\cdot\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -\boldsymbol{T}\cdot\frac{\mathrm{d}}{\mathrm{d}s}\left(\frac{\boldsymbol{G}}{|\boldsymbol{G}|}\right) = -\boldsymbol{T}\cdot\left(\frac{\dfrac{\mathrm{d}\boldsymbol{G}}{\mathrm{d}s}|\boldsymbol{G}| - \boldsymbol{G}\dfrac{\mathrm{d}|\boldsymbol{G}|}{\mathrm{d}s}}{|\boldsymbol{G}|^2}\right) = -\boldsymbol{T}\cdot\frac{\dfrac{\mathrm{d}\boldsymbol{G}}{\mathrm{d}s}}{|\boldsymbol{G}|}\\
&= -\frac{\boldsymbol{T}}{|\boldsymbol{G}|}\cdot\left(\frac{\partial\boldsymbol{G}}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}s} + \frac{\partial\boldsymbol{G}}{\partial v}\frac{\mathrm{d}v}{\mathrm{d}s}\right) = -\frac{\boldsymbol{R}_{a,b}}{|\boldsymbol{R}_{a,b}|^2|\boldsymbol{G}|}\cdot\left(a\frac{\partial\boldsymbol{G}}{\partial u} + b\frac{\partial\boldsymbol{G}}{\partial v}\right)\\
&= -\frac{a\boldsymbol{p}_u + b\boldsymbol{p}_v}{|\boldsymbol{R}_{a,b}|^2|\boldsymbol{G}|}\cdot\left[a(\boldsymbol{p}_{uu}\times\boldsymbol{p}_v + \boldsymbol{p}_u\times\boldsymbol{p}_{uv}) + b(\boldsymbol{p}_{uv}\times\boldsymbol{p}_v + \boldsymbol{p}_u\times\boldsymbol{p}_{vv})\right]\\
&\qquad (\text{note } (\boldsymbol{v}_1\times\boldsymbol{v}_2)\cdot\boldsymbol{v}_3 = (\boldsymbol{v}_2\times\boldsymbol{v}_3)\cdot\boldsymbol{v}_1 = -(\boldsymbol{v}_3\times\boldsymbol{v}_2)\cdot\boldsymbol{v}_1)\\
&= \frac{a^2(\boldsymbol{p}_u\times\boldsymbol{p}_v)\cdot\boldsymbol{p}_{uu} + 2ab(\boldsymbol{p}_u\times\boldsymbol{p}_v)\cdot\boldsymbol{p}_{uv} + b^2(\boldsymbol{p}_u\times\boldsymbol{p}_v)\cdot\boldsymbol{p}_{vv}}{|\boldsymbol{R}_{a,b}|^2|\boldsymbol{G}|}\\
&= \frac{1}{|\boldsymbol{R}_{a,b}|^2}\begin{bmatrix} a & b\end{bmatrix}\begin{bmatrix}\boldsymbol{N}\cdot\boldsymbol{p}_{uu} & \boldsymbol{N}\cdot\boldsymbol{p}_{uv}\\ \boldsymbol{N}\cdot\boldsymbol{p}_{uv} & \boldsymbol{N}\cdot\boldsymbol{p}_{vv}\end{bmatrix}\begin{bmatrix} a\\ b\end{bmatrix}\\
&= \frac{1}{|\boldsymbol{R}_{a,b}|^2}\begin{bmatrix} a & b\end{bmatrix}\left(-\begin{bmatrix}\boldsymbol{p}_u^{\mathrm{T}}\\ \boldsymbol{p}_v^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\boldsymbol{N}_u & \boldsymbol{N}_v\end{bmatrix}\right)\begin{bmatrix} a\\ b\end{bmatrix},\\
&\qquad (\text{note } \boldsymbol{N}_u\cdot\boldsymbol{p}_u + \boldsymbol{N}\cdot\boldsymbol{p}_{uu} = (\boldsymbol{N}\cdot\boldsymbol{p}_u)_u = 0)
\end{aligned}
$$

where the matrix $\boldsymbol{J}_{\mathrm{II}} \equiv -\begin{bmatrix}\boldsymbol{p}_u & \boldsymbol{p}_v\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\boldsymbol{N}_u & \boldsymbol{N}_v\end{bmatrix}$ is called the *second fundamental form*.

Denote $\boldsymbol{v}_{a,b} \equiv \begin{bmatrix} a & b\end{bmatrix}^{\mathrm{T}}$ which can be an arbitrary non-zero vector and the curvature $k_{a,b}$ is

$$k_{a,b} = \frac{\boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{II}}\boldsymbol{v}_{a,b}}{\boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b}}. \tag{D.18}$$

Denote the surface's **principal curvatures** as $k_{\min}$ and $k_{\max}$, namely $k_{\min} \leqslant k_{a,b} \leqslant k_{\max}$; in fact, the principal curvatures $k_{\min}$ and $k_{\max}$ are the two eigenvalues of $\boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1}$ or $\boldsymbol{J}_{\mathrm{I}}^{-1}\boldsymbol{J}_{\mathrm{II}}$. To understand this, first note that (D.18) is equivalent to

$$k_{a,b} = \boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{II}}\boldsymbol{v}_{a,b} \quad (s.t. \quad \boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b} = 1) \tag{D.19}$$

and then construct a *Lagrange multiplier* [17]

$$L(\boldsymbol{v}_{a,b}, \lambda) = \boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{II}}\boldsymbol{v}_{a,b} - \lambda(\boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b} - 1),$$

which gives the (first) optimality condition

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{v}_{a,b}} & = 2(\boldsymbol{J}_{\mathrm{II}} - \lambda \boldsymbol{J}_{\mathrm{I}})\boldsymbol{v}_{a,b} = 2(\boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1} - \lambda \boldsymbol{I})\boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b} = 0 \\ \frac{\partial L}{\partial \lambda} & = \boldsymbol{v}_{a,b}^{\mathrm{T}}\boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b} - 1 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} \lambda & \text{is an eigenvalue of } \boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1} \\ \boldsymbol{J}_{\mathrm{I}}\boldsymbol{v}_{a,b} & \text{is non} - \text{zero and is an eigenvector of } \boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1} \end{cases}$$

Since $k_{a,b}$ in (D.19) is bounded and has a minimum value and a maximum value apparently, the two eigenvalues obtained by solving above optimality condition right correspond to the minimum curvature $k_{\min}$ and the maximum curvature $k_{\max}$.

The **mean curvature** k_M and the **Gaussian curvature** k_G are defined respectively as

$$k_M = \frac{k_{\min}+k_{\max}}{2} = \frac{1}{2}\mathrm{tr}(\boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1}), \quad k_G = k_{\min}k_{\max} = \det\,(\boldsymbol{J}_{\mathrm{II}}\boldsymbol{J}_{\mathrm{I}}^{-1}). \qquad \text{(D.20)}$$

D.2.3 Curvature Formulas for Implicit Surfaces

Given a generic point on a surface represented implicitly by (D.7), its normal vector is the gradient

$$\nabla F \equiv [F_x \quad F_y \quad F_z]^{\mathrm{T}}, \text{i. e.} \begin{bmatrix} \frac{\partial \mathrm{F}}{\partial \mathrm{x}} & \frac{\partial \mathrm{F}}{\partial \mathrm{y}} & \frac{\partial \mathrm{F}}{\partial \mathrm{z}} \end{bmatrix}^{\mathrm{T}}$$

and its unit normal vector is $\boldsymbol{N} = \nabla F / |\nabla F|$. Consider a generic unit tangent vector $\boldsymbol{T}$ and the curvature $k_{\boldsymbol{T}}$ associated with $\boldsymbol{T}$. The Frenet formula gives

$$\begin{aligned} k_{\boldsymbol{T}} &= -\boldsymbol{T} \cdot \frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -\boldsymbol{T} \cdot ((\nabla \boldsymbol{N})\boldsymbol{T}) = -\boldsymbol{T}^{\mathrm{T}}\nabla\left(\frac{\nabla F}{|\nabla F|}\right)\boldsymbol{T} \\ &= -\boldsymbol{T}^{\mathrm{T}}\frac{\nabla^2 F|\nabla F| - \nabla F \nabla |\nabla F|}{|\nabla F|^2}\boldsymbol{T} \\ &= -\frac{\boldsymbol{T}^{\mathrm{T}}\nabla^2 F\boldsymbol{T}}{|\nabla F|} \quad (s.t. \quad \boldsymbol{N}^{\mathrm{T}}\boldsymbol{T} = 0 \,\text{and}\, \mathbf{T}^{\mathrm{T}}\mathbf{T} = 1) \end{aligned} \qquad \text{(D.21)}$$

where $\nabla^2 F$ denotes the Hessian matrix.

Suppose $k_{\min} \leqslant k_{\boldsymbol{T}} \leqslant k_{\max}$. To derive the principal curvatures $k_{\min}$ and $k_{\max}$, construct a Lagrange multiplier

$$L(\boldsymbol{T}, \lambda, \mu) = \frac{1}{2}\boldsymbol{T}^{\mathrm{T}}\nabla^2 F\boldsymbol{T} - \frac{\lambda}{2}(\boldsymbol{T}^{\mathrm{T}}\boldsymbol{T} - 1) - \mu\boldsymbol{N}^{\mathrm{T}}\boldsymbol{T},$$

which gives the (first) optimality condition

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{T}} & = \nabla^2 F\boldsymbol{T} - \lambda\boldsymbol{T} - \mu\boldsymbol{N} = 0 \\ {[9pt]}\frac{\partial L}{\partial \lambda} & = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{T} - 1 = 0 \\ {[9pt]}\frac{\partial L}{\partial \mu} & = \boldsymbol{N}^{\mathrm{T}}\boldsymbol{T} = 0 \end{cases} \tag{D.22}$$

The first and third equations of the equation group (D.22) are equivalent to

$$\begin{bmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \boldsymbol{N} \\ \boldsymbol{N}^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{T} \\ -\mu \end{bmatrix} = 0 \Longleftrightarrow \begin{bmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{T} \\ -\frac{\mu}{|\nabla F|} \end{bmatrix} = 0, \tag{D.23}$$

where $[\boldsymbol{T} \quad -\mu/|\nabla F|]^{\mathrm{T}}$ must be a non-trivial solution because $\boldsymbol{T}^{\mathrm{T}}\boldsymbol{T} = 1$, i.e. the second equation of (D.22) and $\boldsymbol{T} \neq 0$. So

$$D(\lambda) \equiv \begin{vmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{vmatrix} = \det\left(\begin{bmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{bmatrix}\right) = 0. \tag{D.24}$$

Note

$$\begin{aligned} & \begin{bmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} -(\nabla^2 F - \lambda\boldsymbol{I})^{-1}\nabla F & \boldsymbol{I} \\ 1 & 0 \end{bmatrix} \\ = & \begin{bmatrix} 0 & \nabla^2 F - \lambda\boldsymbol{I} \\ -\nabla F^{\mathrm{T}}(\nabla^2 F - \lambda\boldsymbol{I})^{-1}\nabla F & \nabla F^{\mathrm{T}} \end{bmatrix} \\ \Rightarrow & \begin{vmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{vmatrix}(-1) = |\nabla^2 F - \lambda\boldsymbol{I}|\nabla F^{\mathrm{T}}(\nabla^2 F - \lambda\boldsymbol{I})^{-1}\nabla F \\ \Rightarrow & \, D(\lambda) = -\nabla F^{\mathrm{T}}(\nabla^2 F - \lambda\boldsymbol{I})^{*}\nabla F, \end{aligned}$$

where $(\nabla^2 F - \lambda\boldsymbol{I})^{*}$ is the *adjoint matrix* of $\nabla^2 F - \lambda\boldsymbol{I}$. Apparently, $D(\lambda)$ is a polynomial of order no more than two; let $D(\lambda) \equiv c_2\lambda^2 + c_1\lambda + c_0$. Further obtain

$$c_0 \quad = D(0) = \begin{vmatrix} \nabla^2 F & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{vmatrix}, \tag{D.25a}$$

$$c_2 \quad = \lim_{\lambda\to\infty} \frac{D(\lambda)}{\lambda^2} = \lim_{\lambda\to\infty} \frac{-\nabla F^{\mathrm{T}}(-\lambda\boldsymbol{I})^{*}\nabla F}{\lambda^2} = -\nabla F^{\mathrm{T}}\nabla F = -|\nabla F|^2, \tag{D.25b}$$

$$\begin{aligned} c_1 &= -\nabla F^{\mathrm{T}} \left\{ \lambda - \text{coefficients of } \begin{bmatrix} F_{xx}-\lambda & F_{xy} & F_{xz} \\ F_{xy} & F_{yy}-\lambda & F_{yz} \\ F_{xz} & F_{yz} & F_{zz}-\lambda \end{bmatrix}^* \right\} \nabla F \\ &= -\nabla F^{\mathrm{T}} \begin{bmatrix} -F_{yy}-F_{zz} & F_{xy} & F_{xz} \\ F_{xy} & -F_{xx}-F_{zz} & F_{yz} \\ F_{xz} & F_{yz} & -F_{xx}-F_{yy} \end{bmatrix} \nabla F \\ &= \nabla F^{\mathrm{T}} (\mathrm{tr}(\nabla^2 F)\boldsymbol{I} - \nabla^2 F)\nabla F = |\nabla F|^2 \mathrm{tr}(\nabla^2 F) - \nabla F^{\mathrm{T}} \nabla^2 F \nabla F. \end{aligned} \tag{D.25c}$$

Substitute (D.22) into (D.21) and obtain

$$k_{\boldsymbol{T}} = -\frac{\boldsymbol{T}^{\mathrm{T}}\nabla^2 F\boldsymbol{T}}{|\nabla F|} = -\frac{\boldsymbol{T}^{\mathrm{T}}(\lambda\boldsymbol{T}+\mu\boldsymbol{N})}{|\nabla F|} = -\frac{\lambda\boldsymbol{T}^{\mathrm{T}}\boldsymbol{T}}{|\nabla F|} = -\frac{\lambda}{|\nabla F|}. \tag{D.26}$$

Since $k_{\boldsymbol{T}}$ in (D.21) is bounded and has a minimum value and a maximum value apparently, the two roots of $D(\lambda)$ correspond right to the minimum curvature $k_{\min}$ and the maximum curvature $k_{\max}$, namely

$$\{k_{\min}, k_{\max}\} = -\frac{\text{roots of } \left\{ \begin{vmatrix} \nabla^2 F - \lambda\boldsymbol{I} & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{vmatrix} \right\}}{|\nabla F|}. \tag{D.27}$$

The **mean curvature** k_M and the **Gaussian curvature** k_G are computed respectively as

$$k_M = \frac{k_{\min}+k_{\max}}{2} = -\frac{-\dfrac{c_1}{2c_2}}{|\nabla F|} = \frac{\nabla F^{\mathrm{T}}\nabla^2 F\nabla F - |\nabla F|^2 \mathrm{tr}(\nabla^2 F)}{2|\nabla F|^3}, \tag{D.28a}$$

$$k_G = k_{\min}k_{\max} = \frac{\dfrac{c_0}{c_2}}{|\nabla F|^2} = -\frac{\begin{vmatrix} \nabla^2 F & \nabla F \\ \nabla F^{\mathrm{T}} & 0 \end{vmatrix}}{|\nabla F|^4} = \frac{\nabla F^{\mathrm{T}}\nabla^2 F^* \nabla F}{|\nabla F|^4}. \tag{D.28b}$$

For example, consider an implicit sphere surface $F(x, y, z) = x^2 + y^2 + z^2 = R^2$ which has

$$\nabla F = \begin{bmatrix} 2x \\ 2y \\ 2z \end{bmatrix}, \quad \nabla^2 F = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

Apply (D.28) and obtain

$$k_M = \frac{\begin{bmatrix}2x & 2y & 2z\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 2\end{bmatrix}\begin{bmatrix}2x\\2y\\2z\end{bmatrix} - \left(2\sqrt{x^2+y^2+z^2}\right)^2 \mathrm{tr}\left(\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 2\end{bmatrix}\right)}{2\left(2\sqrt{x^2+y^2+z^2}\right)^3} = -\frac{1}{R},$$

$$k_G = -\frac{\begin{vmatrix}2 & 0 & 0 & 2x\\0 & 2 & 0 & 2y\\0 & 0 & 2 & 2z\\2x & 2y & 2z & 0\end{vmatrix}}{\left(2\sqrt{x^2+y^2+z^2}\right)^4} = \frac{\begin{bmatrix}2x & 2y & 2z\end{bmatrix}\begin{bmatrix}4 & 0 & 0\\0 & 4 & 0\\0 & 0 & 4\end{bmatrix}\begin{bmatrix}2x\\2y\\2z\end{bmatrix}}{16(x^2+y^2+z^2)^2} = \frac{1}{R^2},$$

which is consistent with the common sense that a sphere surface of radius R has an unsigned curvature $1/R$ everywhere.

Footnotes

[8] Since $\boldsymbol{N}\cdot\boldsymbol{N} = 1 \Rightarrow 0 = \frac{\mathrm{d}}{\mathrm{d}s}(\boldsymbol{N}\cdot\boldsymbol{N}) = 2\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s}\cdot\boldsymbol{N}$ namely $\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \perp \boldsymbol{N}$ and since $\boldsymbol{T} \perp \boldsymbol{N}$, one has

$$\frac{\mathrm{d}\boldsymbol{B}}{\mathrm{d}s} = \frac{\mathrm{d}}{\mathrm{d}s}(\boldsymbol{T}\times\boldsymbol{N}) = \frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s}\times\boldsymbol{N} + \boldsymbol{T}\times\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = k\boldsymbol{N}\times\boldsymbol{N} + \boldsymbol{T}\times\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = \boldsymbol{T}\times\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} \propto \boldsymbol{N},$$

which explains validity of the torsion definition in (D.8c). Besides,

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} &= \boldsymbol{T}\boldsymbol{T}^{\mathrm{T}}\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} + \left(|\boldsymbol{T}|^2\boldsymbol{I} - \boldsymbol{T}\boldsymbol{T}^{\mathrm{T}}\right)\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -k\boldsymbol{T} - \boldsymbol{S}_{\boldsymbol{T}}^2\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -k\boldsymbol{T} - \boldsymbol{T}\times\left(\boldsymbol{T}\times\frac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s}\right)\\ &= -k\boldsymbol{T} - \boldsymbol{T}\times\frac{\mathrm{d}}{\mathrm{d}s}(\boldsymbol{T}\times\boldsymbol{N}) = -k\boldsymbol{T} - \boldsymbol{T}\times(-\tau\boldsymbol{N}) = -k\boldsymbol{T} + \tau(\boldsymbol{T}\times\boldsymbol{N}) = -k\boldsymbol{T} + \tau\boldsymbol{B},\end{aligned}$$

which gives derivation of (D.8b). Some details in (D.10) and (D.14) are used in above derivation.

[9] Given a vector $\boldsymbol{v} = \begin{bmatrix}v_1 & v_2 & v_3\end{bmatrix}^{\mathrm{T}}$, define its skew-symmetric matrix as

$$\boldsymbol{S}_{\boldsymbol{v}} \equiv \begin{bmatrix}0 & -v_3 & v_2\\v_3 & 0 & -v_1\\-v_2 & v_1 & 0\end{bmatrix},$$

which satisfies $\boldsymbol{S}_{\boldsymbol{v}}^{\mathrm{T}} = -\boldsymbol{S}_{\boldsymbol{v}}$ and $\boldsymbol{S}_{\boldsymbol{v}}^3 = -|\boldsymbol{v}|^2\boldsymbol{S}_{\boldsymbol{v}}$. Inversely, given a skew-symmetric matrix $\boldsymbol{W}$, define its associated vector as $\boldsymbol{v} \equiv \boldsymbol{S}_{\boldsymbol{W}}^{[-1]}$ such that $\boldsymbol{W} = \boldsymbol{S}_{\boldsymbol{v}}$.

[10] $< \boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n >$ denotes the sub-space spanned by the vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, ..., $\boldsymbol{v}_n$.

Appendix E

Scale-Space Theory

Raw images are rarely used directly for image processing in practical applications. A preliminary step of image filtering is normally involved, motivation for which has been clarified in Appendix C. As analysed theoretically, the Wiener filter is the optimal filter in a rather wide sense. Despite its theoretical importance, the Wiener filter is usually not an “optimal” choice or even a feasible choice in practice, especially when *a priori* knowledge of imaging statistics (including those of the vision system as well as those of the perceived entities) is lacking.

In practical applications, a “good-enough” **parametric filter** , instead of the theoretically-optimal Wiener filter, tends to be adopted, with the filter parameter(s) tuned empirically or according to certain adaptive conditions. A parametric filter’s parameter or dominant parameter is normally associated with the **filtering scale**, which roughly determines the range of a pixel’s neighbourhood that mainly influences the pixel’s filtered result. So for simplicity yet without loss of generality, the author only considers one-parameter (i.e. the filtering-scale parameter) filter here. For example, the Gaussian filter presented in Sect. C.4.2 in Appendix C is a representative one-parameter filter .[11] which is an appropriate approximation of the Wiener filter. In fact, as will be clarified latter in this appendix, the Gaussian filter also plays an indispensable role in scale-space theory.

A potential way of using a parametric filter is to set the filter parameter empirically or adaptively, which is acceptable in practical applications where sufficient *a priori* knowledge is available. On the other hand, in many practical applications, how to set the filter parameter properly is not a trivial problem.

Instead of setting the filter parameter empirically or adaptively, an advanced methodology of using a parametric filter is to analyse how a series of filter parameters influence the image and to extract image features that are salient and stable across filtered images associated with various filter parameters. This idea motivates the development of the scale-space theory [18, 19]. As explained above, the author only considers the filtering-scale parameter here.

Given an image and the image pixel value function denoted as $I(u, v)$ ($1 \leqslant u \leqslant W$, $1 \leqslant v \leqslant H$), where W and H denote the image width and height respectively. Each filtering scale corresponds to a specific filtered image. All the filtered images form the original image’s **scale space**, which can be represented by a function $I_{\mathrm{F}}(u, v, w)$ defined over the trivariate domain $\{(u, v, w) \mid 1 \leqslant u \leqslant W, 1 \leqslant v \leqslant H, w \geqslant 0\}$. The trivariate function $I_{\mathrm{F}}(u, v, w)$ is called the **filtered image function** , where u and v still denote the image coordinates. The function image of $I_{\mathrm{F}}(u, v, w)$ ($1 \leqslant u \leqslant W$, $1 \leqslant v \leqslant H$), or simply $I_{\mathrm{F}}(u, v, w)$ without (u, v) domain specification[12] , denotes the filtered image associated with the filtering scale *w*. Besides, $I_{\mathrm{F}}(u,v,0)$ denotes the original image without filtering.

To judge whether a parametric filter can properly influence the image as the filtering scale increases, the **causality** criterion introduced by [18, 20] is adopted: any feature associated with a large filtering scale (i.e. at a coarse level of resolution) must continuously possess at least a “cause” feature associated with a small filtering scale (i.e. at a fine level of resolution). In other words, any feature associated with any filtering scale must stem continuously from at least an ultimate “cause” feature in the original image. The causality criterion requires that the parametric filter must not let the image generate extra features, i.e. fake features as the filtering scale increases. Therefore, following the causality criterion, a generic surface $I_{\mathrm{F}}(u,v,w)=I_0$ is of a roughly concave shape with the cave mouth pointing towards the smaller filtering scale side, as illustrated in Fig. E.1.

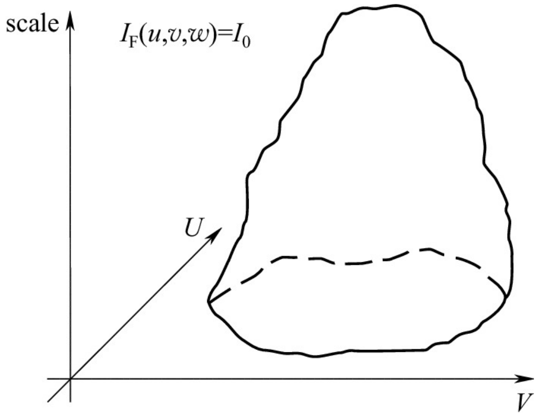


Fig. E.1 Generic surface in the scale space

Consider extremum points (also known as stationary points) of an image because they are salient features that can fairly characterize inherent structures of the image. The gradient at a generic point of the surface $I_{\mathrm{F}}(u,v,w)=I_0$ is

$$\nabla I_{\mathrm{F}} = \begin{bmatrix} I_{\mathrm{Fu}} & I_{\mathrm{Fv}} & I_{\mathrm{Fw}} \end{bmatrix}^{\mathrm{T}} \equiv \begin{bmatrix} \frac{\partial I_{\mathrm{F}}}{\partial u} & \frac{\partial I_{\mathrm{F}}}{\partial v} & \frac{\partial I_{\mathrm{F}}}{\partial w} \end{bmatrix}^{\mathrm{T}}$$

which is a normal vector at the point. At an extremum point (by default, suppose it is non-degenerate), $I_{\mathrm{Fu}} = I_{\mathrm{Fv}} = 0$, and $\nabla I_{\mathrm{F}} = \begin{bmatrix} 0 & 0 & I_{\mathrm{Fw}} \end{bmatrix}^{\mathrm{T}}$ is perpendicular to the (*u*, *v*)-plane. According to (D.27), the minimum curvature $k_{\min}$ and the maximum curvature $k_{\max}$ at the extremum point are

$$\begin{aligned} \{k_{\min}, k_{\max}\} &= -\frac{\text{roots of }\left\{\begin{vmatrix} \nabla^2 I_{\mathrm{F}} - \lambda \boldsymbol{I} & \nabla I_{\mathrm{F}} \\ \nabla I_{\mathrm{F}}^{\mathrm{T}} & 0 \end{vmatrix}\right\}}{|\nabla I_{\mathrm{F}}|} \\ &= -\frac{\text{roots of }\{\lambda^2 - (I_{\mathrm{Fuu}} + I_{\mathrm{Fvv}})\lambda + I_{\mathrm{Fuu}} I_{\mathrm{Fvv}} - I_{\mathrm{Fuv}}^2\}}{|I_{\mathrm{Fw}}|}, \end{aligned} \tag{E.1}$$

where $I_{\mathrm{Fuu}}I_{\mathrm{Fvv}} - I_{\mathrm{Fuv}}^2$ should be positive.

According to the Frenet formula (D.8), one has

$$\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = k\boldsymbol{N} = \begin{bmatrix} 0 & 0 & k\frac{I_{\mathrm{Fw}}}{|I_{\mathrm{Fw}}|} \end{bmatrix}^{\mathrm{T}}.$$

As clarified previously, the surface $I_{\mathrm{F}}(u,v,w) = I_0$ is concave towards the downside, so $\frac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s}$ at the extremum point is pointing towards the downside, which implies that $k\frac{I_{\mathrm{Fw}}}{|I_{\mathrm{Fw}}|} < 0$ for both the minimum curvature $k_{\min}$ and the maximum curvature $k_{\max}$ and further implies that $I_{\mathrm{Fuu}} + I_{\mathrm{Fvv}} = -(k_{\min} + k_{\max})|I_{\mathrm{Fw}}|$ and I_{Fw} are of the same sign. More specifically,

$$\frac{I_{\mathrm{Fuu}}+I_{\mathrm{Fvv}}}{I_{\mathrm{Fw}}} = |k_{\min}| + |k_{\max}| > 0. \tag{E.2}$$

Follow the **homogeneity** and **isotropy** criteria proposed by [20]. The meanings of the criteria are two-folds: First, neither point nor direction of the original image should have any speciality *a priori*.[13] Second, the parametric filter should not have special influence on any specific point or direction of the image. Simply speaking, any point or direction of the image should neither have any speciality itself nor have any speciality of how it is treated by the parametric filter . Besides, there is no *a priori* knowledge of the extremum point and it can be anywhere in the image.

Therefore, the curvature sum $|k_{\min}|+|k_{\max}|$ in (E.2), which reflects an inherent property of the filtered image function $I_{\mathrm{F}}(u,v,w)$ at a generic extremum point, should be determined only by the filtering scale *w*, irrelevant to *u* and *v*. Then

$$\frac{I_{\mathrm{Fuu}}+I_{\mathrm{Fvv}}}{I_{\mathrm{Fw}}} = |k_{\min}| + |k_{\max}| \equiv \phi(w) > 0 \Longleftrightarrow I_{\mathrm{Fuu}} + I_{\mathrm{Fvv}} = \phi(w) I_{\mathrm{Fw}}. \tag{E.3}$$

Define a new filtering scale σ via the following transformation

$$\sigma = \int_0^w \frac{1}{\phi(w)} \mathrm{d}w \equiv \varphi(w), \tag{E.4}$$

which describes a bijective mapping between σ and *w* because $\phi(w) > 0$.

Substitute (E.4) into the filtered image function $I_{\mathrm{F}}(u,v,w)$ to obtain a new filtered image function $I_{\mathrm{F}}^{\mathrm{N}}(u,v,\sigma) \equiv I_{\mathrm{F}}(u,v,\varphi^{-1}(\sigma)) = I_{\mathrm{F}}(u,v,w)$. One has

$$I_{\mathrm{F}\sigma}^{\mathrm{N}} \equiv \frac{\partial I_{\mathrm{F}}^{\mathrm{N}}}{\partial \sigma} = \frac{\partial I_{\mathrm{F}}}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}\sigma} = I_{\mathrm{Fw}}\phi(w),$$

so (E.3) becomes

$$\Delta I_{\mathrm{F}}^{\mathrm{N}} \equiv I_{\mathrm{Fuu}}^{\mathrm{N}} + I_{\mathrm{Fvv}}^{\mathrm{N}} = I_{\mathrm{F\sigma}}^{\mathrm{N}}, \quad \text{i.e.} \quad \frac{\partial^2 \mathrm{I}_{\mathrm{F}}^{\mathrm{N}}}{\partial \mathrm{u}^2} + \frac{\partial^2 \mathrm{I}_{\mathrm{F}}^{\mathrm{N}}}{\partial \mathrm{v}^2} = \frac{\partial \mathrm{I}_{\mathrm{F}}^{\mathrm{N}}}{\partial \sigma}, \tag{E.5}$$

where the **Laplacian operator** Δ is taken with respect to the image coordinates (u, v). The partial differential equation described in (E.5) is the famous **heat equation** also known as **diffusion equation** [21].

It is known that the unnormalized Gaussian function

$$g(u, v, \sigma) = \frac{1}{\sigma} \mathrm{e}^{-\frac{u^2+v^2}{4\sigma}} \tag{E.6}$$

satisfies the heat equation (E.5), verification of which can be given by

$$\begin{aligned}
\frac{\partial g}{\partial \sigma} &= g\frac{u^2+v^2}{4\sigma^2} + g(-\tfrac{1}{\sigma}) = g\left(\frac{u^2+v^2}{4\sigma^2} - \frac{1}{\sigma}\right), \\
\frac{\partial g}{\partial u} &= g(-\tfrac{u}{2\sigma}), \quad \frac{\partial g}{\partial v} = g(-\tfrac{v}{2\sigma}), \\
\frac{\partial^2 g}{\partial u^2} &= g(-\tfrac{u}{2\sigma})^2 + g(-\tfrac{1}{2\sigma}) = g\left(\frac{u^2}{4\sigma^2} - \frac{1}{2\sigma}\right), \\
\frac{\partial^2 g}{\partial v^2} &= g(-\tfrac{v}{2\sigma})^2 + g(-\tfrac{1}{2\sigma}) = g\left(\frac{v^2}{4\sigma^2} - \frac{1}{2\sigma}\right),
\end{aligned}$$

so

$$\Delta g = \frac{\partial^2 g}{\partial u^2} + \frac{\partial^2 g}{\partial v^2} = g\left(\frac{u^2+v^2}{4\sigma^2} - \frac{1}{\sigma}\right) = \frac{\partial g}{\partial \sigma}.$$

The Gaussian function (E.6) is called the **fundamental solution** of the heat equation (E.5). Note that the heat equation is linear, and its solution multiplied by a constant factor is still its solution. So the Gaussian function (E.6), be it normalized or not, can always be treated as the fundamental solution of the heat equation. Yet for analysis convenience below, adopt the normalized Gaussian function

$$g(u, v, \sigma) = \frac{1}{4\pi\sigma} \mathrm{e}^{-\frac{u^2+v^2}{4\sigma}} \tag{E.7}$$

as the fundamental solution of the heat equation.

Consider (E.5) under the initial-value condition $I_{\mathrm{F}}^{\mathrm{N}}(u, v, 0) = I(u, v)$. Define two-dimensional (u, v)-convolution based function transformation

$$I_{\mathrm{F}}^{\mathrm{N}}(u, v, \sigma) \equiv g(u, v, \sigma) * h(u, v, \sigma), \tag{E.8}$$

which is normally valid.[14] Note that $\lim_{\sigma \to 0} g(u, v, \sigma) = \delta(u, v)$ where $\delta(\cdot)$ is the *Dirac function*. The initial-value condition $I_{\mathrm{F}}^{\mathrm{N}}(u, v, 0) = I(u, v)$ implies

$$I_{\mathrm{F}}^{\mathrm{N}}(u,v,0)=g(u,v,0)*h(u,v,0)=\delta(u,v)*h(u,v,0)=h(u,v,0)=I(u,v). \tag{E.9}$$

Substitute (E.8) into (E.5) and obtain

$$\begin{aligned}\Delta I_{\mathrm{F}}^{\mathrm{N}} &= \Delta(g*h)=\Delta g*h=\frac{\partial g}{\partial\sigma}*h,\\ \frac{\partial I_{\mathrm{F}}^{\mathrm{N}}}{\partial\sigma} &= \frac{\partial(g*h)}{\partial\sigma}=\frac{\partial g}{\partial\sigma}*h+g*\frac{\partial h}{\partial\sigma},\\ \Delta I_{\mathrm{F}}^{\mathrm{N}} &= \frac{\partial I_{\mathrm{F}}^{\mathrm{N}}}{\partial\sigma}\Longleftrightarrow g*\frac{\partial h}{\partial\sigma}=0\Longleftrightarrow\frac{\partial h}{\partial\sigma}=0.\end{aligned} \tag{E.10}$$

Associate (E.10) with (E.9) and have

$$h(u,v,\sigma)=I(u,v). \tag{E.11}$$

So the solution of the heat equation (E.5) under the initial-value condition $I_{\mathrm{F}}^{\mathrm{N}}(u,v,0)=I(u,v)$ is

$$I_{\mathrm{F}}^{\mathrm{N}}(u,v,\sigma)\equiv g(u,v,\sigma)*h(u,v,\sigma)=g(u,v,\sigma)*I(u,v). \tag{E.12}$$

The result described in (E.12) conveys that the filtered image function $I_{\mathrm{F}}^{\mathrm{N}}(u,v,\sigma)$ is the same to that obtained by performing parametric Gaussian filtering $g(u,v,\sigma)$ on the original image pixel value function $I(u, v)$. Besides, above scale-space analysis also conveys that the filtered image function $I_{\mathrm{F}}^{\mathrm{N}}(u,v,\sigma)$ given in (E.12) is the only solution that satisfies the causality, homogeneity, and isotropy criteria. Therefore, the parametric Gaussian filter is the only choice of parametric filter that satisfies these three fundamental criteria and plays an indispensable role in scale-space theory.

Footnotes

[11] The Gaussian filter commonly adopted in image processing may take two parameters, namely the Gaussian filter standard deviation and the locally-truncated Gaussian filter size demonstrated in code **Gaussian-Filter2D.m**; yet the parameter of standard deviation is the filtering-scale parameter that plays a dominant role in determining the Gaussian filter shape, whereas the parameter of filter size plays a subordinate role and can be fairly set by default according to the parameter of standard deviation. So the Gaussian filter essentially has only one parameter, i.e. the filtering-scale parameter.

[12] Without causing confusion, the function image of $I_{\mathrm{F}}(u,v,w)$ ($1\leqslant u\leqslant W$, $1\leqslant v\leqslant H$) may also be simply denoted as $I_{\mathrm{F}}(u,v,w)$, omitting specification of the (u, v) domain range. Therefore, just like the conventional function notation $f(x)$ which may denote both a specific image value corresponding to certain x and the entire function image of $f(x)$ over x's domain, the notation $I_{\mathrm{F}}(u,v,w)$ may denote both a specific pixel value corresponding to certain (u, v, w) and the entire function image of $I_{\mathrm{F}}(u,v,w)$ over (u, v)'s domain $\{(u,v)\mid 1\leqslant u\leqslant W,1\leqslant v\leqslant H\}$.

[13] This does not mean at all that all image points should have the same pixel value. This may be understood from the probabilistic perspective, namely all image points have the same *a priori* statistics.

The probabilistic expectations at any point and along any direction should be the same.

[14] An analysis not so rigorous is given below: Denote $G \equiv F\{g\}$, namely the Gaussian function g's two-dimensional (u, v)-Fourier transform which exists. Denote $g' \equiv F^{-1}\left\{\frac{1}{G}\right\}$, then $g * g' = F^{-1}\left\{G\frac{1}{G}\right\} = F^{-1}\{1\} = \delta(\cdot)$. Let $h \equiv g' * I_{\mathrm{F}}^{\mathrm{N}}$, one has $g * h = g * (g' * I_{\mathrm{F}}^{\mathrm{N}}) = (g * g') * I_{\mathrm{F}}^{\mathrm{N}} = \delta * I_{\mathrm{F}}^{\mathrm{N}} = I_{\mathrm{F}}^{\mathrm{N}}$.

Appendix F

Equations

Readers had better possess a preliminary knowledge of *linear algebra*, for which one may refer to [22–24].

F.1 Linear Equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$

Consider how to solve the linear equation (group)

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \tag{F.1}$$

Let $\boldsymbol{A}$ and $\boldsymbol{y}$ in (F.1) denote a generic matrix and a generic column vector respectively and suppose they are known. Let $\boldsymbol{x}$ denote the unknown column vector to be solved. Besides, assume the dimensions of the matrix $\boldsymbol{A}$, the column vector $\boldsymbol{x}$, and the column vector $\boldsymbol{y}$ are consistent with each other.

Case 1: If the matrix $\boldsymbol{A}$ is square and invertible, then the solution of $\boldsymbol{x}$ is uniquely determined as

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}, \tag{F.2}$$

which can be obtained via the *Gaussian elimination method* [25, 26].

Case 2: If the matrix $\boldsymbol{A}$ is rank-deficient—for example, $\boldsymbol{A}$ is a “fat” matrix (i.e. a matrix with less rows than columns) and hence is definitely rank-deficient—then (F.1) has either an infinite number of $\boldsymbol{x}$ solutions or no solution.[15]

Case 3: If the matrix $\boldsymbol{A}$ is a “thin” matrix (i.e. a matrix with more rows than columns) of full rank and the augmented matrix [22]

$$[\boldsymbol{A} \quad \boldsymbol{y}]$$

has a rank the same to the dimension of $\boldsymbol{x}$ (in other words, the equations are consistent), then (F.1) has a unique solution of $\boldsymbol{x}$ which can also be obtained via the *Gaussian elimination method*

Case 4: If the matrix $\boldsymbol{A}$ is a “thin” matrix of full rank and the augmented matrix

$$[\boldsymbol{A} \quad \boldsymbol{y}]$$

has a rank one more than the dimension of $\boldsymbol{x}$ (in other words, the equations contain self-contradiction), then (F.1) has no solution of $\boldsymbol{x}$.

In strict mathematical sense, above four cases account for all the story of linear equation (group) solving. However, in practical applications, many linear equations that one may encounter belong to the *Case 4*, but one cannot be satisfied with the simple conclusion that they have no solutions.

For example, suppose there is a vehicle moving at constant speed along a straight road. Besides, suppose there is a sensor to monitor the vehicle position every five minutes since twelve and a half o’clock and denote the vehicle position measurements since then consecutively as 30.3 km, 34.9 km, 40.4 km, 45.1 km, 49.5 km, ..., and so on. Now one wants to know the vehicle position at twelve o’clock and the constant vehicle speed. How to solve the problem?

One may set two unknown variables: p_0 km denotes the vehicle position at twelve o’clock, and v km/minute denotes the constant vehicle speed. Then one can establish a group of equations in terms of p_0 and v according to vehicle position measurements:

$$\begin{aligned} p_0 + 30v &= 30.3 \\ p_0 + 35v &= 34.9 \\ p_0 + 40v &= 40.4 \\ p_0 + 45v &= 45.1 \\ p_0 + 50v &= 49.5 \\ \vdots \quad & \quad \vdots \end{aligned} \tag{F.3}$$

or in matrix form as

$$\begin{matrix} 1 & 30 \\[-2pt] 1 & 35 \\[-2pt] 1 & 40 \\[-2pt] 1 & 45 \\[-2pt] 1 & 50 \\[-2pt] \vdots & \vdots \end{matrix} \begin{bmatrix} p_0 \\ v \end{bmatrix} = \begin{matrix} 30.3 \\[-2pt] 34.9 \\[-2pt] 40.4 \\[-2pt] 45.1 \\[-2pt] 49.5 \\[-2pt] \vdots \end{matrix} .$$

One can verify

$$\text{rank} \begin{matrix} 1 & 30 \\[-2pt] 1 & 35 \\[-2pt] 1 & 40 \\[-2pt] 1 & 45 \\[-2pt] 1 & 50 \\[-2pt] \vdots & \vdots \end{matrix} = 2, \quad \text{rank} \begin{matrix} 1 & 30 & 30.3 \\[-2pt] 1 & 35 & 34.9 \\[-2pt] 1 & 40 & 40.4 \\[-2pt] 1 & 45 & 45.1 \\[-2pt] 1 & 50 & 49.5 \\[-2pt] \vdots & \vdots & \vdots \end{matrix} = 3,$$

which implies that (F.3) has no solution—In fact, one can know this conclusion simply by observing self-contradiction among the first three equations—The group of equations in (F.3) demonstrate an example of the *Case 4* that one may encounter in practical applications.

Obviously one cannot be satisfied with the simple conclusion that (F.3) has no solution, because the vehicle itself as well as its constant-velocity motion exists objectively and there must be a solution of p_0 and *v*. Then why does one get such tricky conclusion that no solution of p_0 and *v* exists?

The reason lies right in noises and errors that exist inevitably in sensor measurements. What if the vehicle position measurements $30.3\ \text{km}$, $34.9\ \text{km}$, $40.4\ \text{km}$, $45.1\ \text{km}$, $49.5\ \text{km}$, ... have sensing errors of $0.3\ \text{km}$, $-0.1\ \text{km}$, $0.4\ \text{km}$, $0.1\ \text{km}$, $-0.5\ \text{km}$, ... respectively and one knows these sensing errors? Then one can know that the vehicle positions at corresponding moments are actually $30.0\ \text{km}$, $35.0\ \text{km}$, $40.0\ \text{km}$, $45.0\ \text{km}$, $50.0\ \text{km}$, ... obtained by correcting the original vehicle position measurements according to their errors respectively. Substitute the corrected vehicle positions namely vehicle position ground-truth into (F.3) and obtain

$$\begin{aligned}
&p_0 + 30v && = 30.0 \\
&{[-2pt]}p_0 + 35v && = 35.0 \\
&{[-2pt]}p_0 + 40v && = 40.0 \\
&{[-2pt]}p_0 + 45v && = 45.0 \\
&{[-2pt]}p_0 + 50v && = 50.0 \\
&{[-2pt]} \qquad \vdots && \qquad \vdots
\end{aligned} \tag{F.4}$$

which clearly has a solution of p_0 and *v*, namely $p_0 = 0$ km, and $v = 1$ km/minute.

However, one can never know sensing errors of vehicle position measurements *a priori*, then how to arrive at the corrected equation version (F.4) that gives a meaningful solution? Unfortunately, there is no way.

Since linear equations of the *Case 4* are often encountered in practical applications due to inevitable noises and errors, to handle the dilemma that they have no solutions in strict mathematical sense but they must have a corresponding meaningful solution in reality, one can interpret the generic linear equation (F.1) from a new perspective:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \Longleftrightarrow \boldsymbol{x} = \text{argmin}_{\boldsymbol{x}}\ \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2, \tag{F.5}$$

where $\|\cdot\|$ denotes certain norm. In other words, solving (F.1) is interpreted as searching for the optimal $\boldsymbol{x}$ that minimizes the overall difference between the vectors $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{y}$.

A conventional choice of the norm in (F.5) is the L_2-norm, following which the formalism (F.5) becomes

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \Longleftrightarrow \boldsymbol{x} = \text{argmin}_{\boldsymbol{x}}\ \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \text{argmin}_{\boldsymbol{x}}\ (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}). \tag{F.6}$$

The optimization problem described in (F.6) is a *quadratic programming* problem, where the quadratic function in terms of $\boldsymbol{x}$ is

$$q(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}}\ \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y}.$$

Compute the gradient of the quadratic function $q(\boldsymbol{x})$:

$$\nabla q(\boldsymbol{x}) = 2\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y},$$

the root of which is

$$\boldsymbol{x} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}. \tag{F.7}$$

The solution given in (F.7) is right the $\boldsymbol{x}$ that minimizes $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ and can be treated as the solution of the linear equation (F.1) of the *Case 4*.

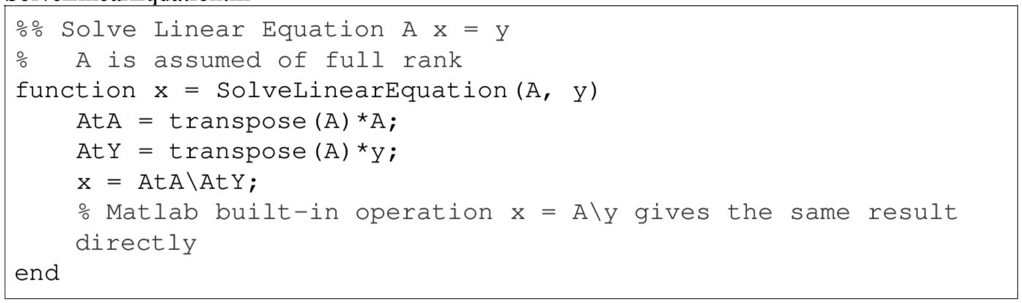
An easy way for readers to memorize the formula (F.7): Left-multiply both sides of (F.1) by $\boldsymbol{A}^{\mathrm{T}}$ and obtain

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \Rightarrow \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y},$$

which belongs to the *Case 1* and has the solution described in (F.7).

Matlab code for solving the generic linear equation (F.1) is given as follows.[16]

SolveLinearEquation.m



```
%% Solve Linear Equation A x = y
%   A is assumed of full rank
function x = SolveLinearEquation(A, y)
    AtA = transpose(A)*A;
    AtY = transpose(A)*y;
    x = AtA\AtY;
    % Matlab built-in operation x = A\y gives the same result
    directly
end
```

F.2 Homogeneous Linear Equation $\boldsymbol{A}\boldsymbol{x} = \mathbf{0}$

Consider how to solve the homogeneous linear equation (group)

$$\boldsymbol{A}\boldsymbol{x} = \mathbf{0}. \tag{F.8}$$

Let $\boldsymbol{A}$ in (F.8) denote a generic matrix that is known. Let $\boldsymbol{x}$ denote the unknown column vector to be solved. Assume the dimensions of the matrix $\boldsymbol{A}$ and the column vector $\boldsymbol{x}$ are consistent with each other.

Some clarifications hover over the homogeneous linear equation (F.8). When a concrete $\boldsymbol{x}$ is a solution of (F.8), any $\lambda\boldsymbol{x}$ that differs from $\boldsymbol{x}$ only by a scaling factor λ is also a solution of (F.8). So people treat solutions that differ from each other only by a scaling factor as a single solution and by convention take their normalized version as the solution.[17]

Another aspect to clarify is that the matrix $\boldsymbol{A}$ encountered in practical applications is often a "thin" matrix (i.e. a matrix with more rows than columns) of full rank, which implies that in strict mathematical sense (F.8) has a trivial solution $\boldsymbol{x} = \mathbf{0}$ only. Like how one interprets the generic linear equation (F.1) from a new perspective in optimization terms, one can also interpret the generic homogeneous linear equation (F.8) from a new perspective in optimization terms:

$$\boldsymbol{A}\boldsymbol{x} = \mathbf{0} \Longleftrightarrow \boldsymbol{x} = \mathrm{argmin}_{\boldsymbol{x}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|^2}{\|\boldsymbol{x}\|^2}, \tag{F.9}$$

where $\|\cdot\|$ denotes certain norm. In other words, solving (F.8) is interpreted as solving the optimization problem described in (F.9).

A conventional choice of the norm in (F.9) is the L_2-norm, following which the formalism (F.9) becomes

$$\boldsymbol{Ax}=\boldsymbol{0}\Longleftrightarrow \boldsymbol{x}=\mathrm{argmin}_{\boldsymbol{x}}\ \frac{\|\boldsymbol{Ax}\|_2^2}{\|\boldsymbol{x}\|_2^2}=\mathrm{argmin}_{\boldsymbol{x}}\ \frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax}}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}. \tag{F.10}$$

The ratio between $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax}$ and $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ in (F.10) is called the *Rayleigh ratio* or *Rayleigh quotient* [27].

Since the matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is real symmetric and positive (semi-)definite, it can be decomposed into

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}=[\boldsymbol{U}_1\quad \boldsymbol{U}_2\quad \cdots\quad \boldsymbol{U}_n]\begin{matrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n\end{matrix}\quad\begin{matrix}\boldsymbol{U}_1^{\mathrm{T}}\\ \boldsymbol{U}_2^{\mathrm{T}}\\ \vdots\\ \boldsymbol{U}_n^{\mathrm{T}}\end{matrix}\equiv\boldsymbol{U\Sigma U}^{\mathrm{T}}, \tag{F.11}$$

where $0\leqslant\sigma_1\leqslant\sigma_2\leqslant\cdots\leqslant\sigma_n$ denote the eigenvalues of $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ [28], and $\boldsymbol{U}$ denotes the orthonormal matrix consisting of the normalized eigenvectors associated with σ_1, σ_2, ..., σ_n respectively. One has

$$\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax}}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}=\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{U\Sigma U}^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{UU}^{\mathrm{T}}\boldsymbol{x}}=\frac{\boldsymbol{x}_{\mathrm{U}}^{\mathrm{T}}\boldsymbol{\Sigma x}_{\mathrm{U}}}{\boldsymbol{x}_{\mathrm{U}}^{\mathrm{T}}\boldsymbol{x}_{\mathrm{U}}},$$

where

$$\boldsymbol{x}_{\mathrm{U}}\equiv\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}\equiv\begin{matrix}\boldsymbol{x}_{\mathrm{U},1}\\ \vdots\\ \boldsymbol{x}_{\mathrm{U},n}\end{matrix}\quad.$$

So

$$\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax}}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}=\frac{\sum_{k=1}^{n}\sigma_k\boldsymbol{x}_{\mathrm{U},k}^2}{\sum_{k=1}^{n}\boldsymbol{x}_{\mathrm{U},k}^2}\geqslant\frac{\sum_{k=1}^{n}\sigma_1\boldsymbol{x}_{\mathrm{U},k}^2}{\sum_{k=1}^{n}\boldsymbol{x}_{\mathrm{U},k}^2}=\sigma_1. \tag{F.12}$$

The equality in (F.12) holds when

$$\boldsymbol{x}_{\mathrm{U},1}=1,\quad \boldsymbol{x}_{\mathrm{U},k}=0\quad(k=2,\cdots,n). \tag{F.13}$$

If the eigenvalues σ_1, σ_2, ..., σ_n are distinct, then the equality in (F.12) holds if and only if the configuration (F.13) is satisfied.

From the configuration (F.13) one can derive

$$\begin{aligned} \boldsymbol{U}^{\mathrm{T}}\boldsymbol{x} &\equiv \begin{matrix} \boldsymbol{x}_{\mathrm{U},1} \\ \vdots \\ \boldsymbol{x}_{\mathrm{U},n} \end{matrix} = \boldsymbol{e}_1 \\ \Rightarrow \quad \boldsymbol{x} &= \boldsymbol{U}\boldsymbol{e}_1 = \boldsymbol{U}_1. \end{aligned} \tag{F.14}$$

So the normalized eigenvector associated with the smallest eigenvalue of $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, as indicated in (F.14), gives the solution of the optimization problem (F.10) and hence gives the solution of the homogeneous linear equation (F.8) as well.

Matlab code for solving the generic homogeneous linear equation (F.8) is given as follows.

SolveHomogeneousLinearEquation.m

```
%% Solve Homogeneous Linear Equation A x = 0
function x = SolveHomogeneousLinearEquation(A)
    AtA = transpose(A)*A;
    [vecE, valE] = eig(AtA); % Matlab built-in function
    valE = diag(valE); mE = min(valE);
    idx = find(valE == mE); idx = idx(1);
    x = vecE(:,idx); % eig gives already normalized eigenvectors
end
```

F.3 Nonlinear Equation $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}$ with Proper Initial Estimate

Consider how to solve the nonlinear equation (group)

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}. \tag{F.15}$$

Let $\boldsymbol{y}$ in (F.15) denote a generic column vector that is known. Let $\boldsymbol{f}(\boldsymbol{x})$ in (F.15) denote a generic nonlinear (continuous) function in terms of $\boldsymbol{x}$. Then $\boldsymbol{f}$ is a n-dimensions-to-m-dimensions mapping

$$\boldsymbol{f} \ : \ \mathbb{R}^n \to \mathbb{R}^m,$$

where n and m denote the dimensions of the unknown vector $\boldsymbol{x}$ and the known vector $\boldsymbol{y}$ respectively.

F.3.1 Perspective of Iterative Linear Equation Solving

Suppose a proper initial estimate of $\boldsymbol{x}$, denoted $\boldsymbol{x}_0$, is available—How the initial estimate $\boldsymbol{x}_0$ is actually obtained depends on field knowledge in practical applications and is out of focus here—The solution of $\boldsymbol{x}$ is refined iteratively from the initial estimate $\boldsymbol{x}_0$. Denote the estimate of $\boldsymbol{x}$ after t rounds of iterative refinement as $\boldsymbol{x}_t$. In other words, from the initial estimate $\boldsymbol{x}_0$, iteratively obtain more and more refined estimates $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, ..., $\boldsymbol{x}_t$, ..., etc.

Given current estimate $\boldsymbol{x}_t$, for sake of obtaining a further refined estimate $\boldsymbol{x}_{t+1}$, perform local linearization of $\boldsymbol{f}(\boldsymbol{x})$ about current estimate $\boldsymbol{x}_t$:

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_t) + \nabla \boldsymbol{f}(\boldsymbol{x}_t)(\boldsymbol{x} - \boldsymbol{x}_t), \tag{F.16}$$

where

$$\nabla \boldsymbol{f}(\boldsymbol{x}_t) \equiv \tfrac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}_t}$$

is the *m*-by-*n* Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$ at $\boldsymbol{x}_t$.

Substitute (F.16) into (F.15) and obtain a linear equation (group)

$$\boldsymbol{f}(\boldsymbol{x}_t) + \nabla \boldsymbol{f}(\boldsymbol{x}_t)(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) = \boldsymbol{y} \Longleftrightarrow \nabla \boldsymbol{f}(\boldsymbol{x}_t)(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t) \tag{F.17}$$

which is a locally approximated version of the original nonlinear equation (F.15). Treat $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ holistically as a variable and $\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t)$ holistically as a constant in (F.17). Solve (F.17) like how to solve the linear equation (F.1).

In practical applications, *m* i.e. the dimension of $\boldsymbol{y}$ is usually much larger than *n* i.e. the dimension of $\boldsymbol{x}$. One usually resorts to (F.7) to solve (F.17) and has

$$\begin{aligned} & \boldsymbol{x}_{t+1} - \boldsymbol{x}_t = \left[\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}_t)\right]^{-1} \nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} [\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t)] \\ \Leftrightarrow \quad & \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \left[\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}_t)\right]^{-1} \nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} [\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t)], \end{aligned} \tag{F.18}$$

which describes how a further refined estimate $\boldsymbol{x}_{t+1}$ is obtained from current estimate $\boldsymbol{x}_t$.

F.3.2 Perspective of Least Mean Squares Optimization

One may interpret the generic nonlinear equation (F.15) from the perspective of least mean squares optimization

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y} \Longleftrightarrow \boldsymbol{x} = \mathrm{argmin}_{\boldsymbol{x}} \ \|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2. \tag{F.19}$$

In other words, solving (F.15) is interpreted as searching for the optimal $\boldsymbol{x}$ that minimizes the overall difference squares[18] between the vectors $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{y}$.

Denote the sum of squares as $q(\boldsymbol{x})$, namely

$$q(\boldsymbol{x}) \equiv \|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 = [\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}]^{\mathrm{T}} [\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}] = \boldsymbol{f}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{f}(\boldsymbol{x}) - 2\boldsymbol{y}^{\mathrm{T}} \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{y}^{\mathrm{T}} \boldsymbol{y}.$$

The gradient of $q(\boldsymbol{x})$ is

$$\nabla q(\boldsymbol{x}) = 2\nabla \boldsymbol{f}(\boldsymbol{x})^{\mathrm{T}} [\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}] \tag{F.20}$$

and the Hessian matrix of $q(\boldsymbol{x})$ is

$$\nabla^2 q(\boldsymbol{x}) = 2\nabla \boldsymbol{f}(\boldsymbol{x})^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}) + 2\textstyle\sum_{i=1}^{n} \left[\frac{\partial}{\partial x_i} \nabla \boldsymbol{f}(\boldsymbol{x})^{\mathrm{T}}\right] [\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}] \boldsymbol{e}_i^{\mathrm{T}}. \tag{F.21}$$

Given a proper initial estimate $\boldsymbol{x}_0$, instead of using (F.18), the solution of $\boldsymbol{x}$ can also be refined iteratively using the *Newton method* [17]

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - [\nabla^2 q(\boldsymbol{x}_t)]^{-1} \nabla q(\boldsymbol{x}_t). \tag{F.22}$$

Besides, usually for a sequence of effectively refined $\{\boldsymbol{x}_t\}$, $\boldsymbol{f}(\boldsymbol{x}_t)$ tends to be close to $\boldsymbol{y}$ or in other words $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}$ tends to be close to zero.[19] So in (F.21), the Hessian matrix can be approximated as

$$\nabla^2 q(\boldsymbol{x}) \approx 2\nabla \boldsymbol{f}(\boldsymbol{x})^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}). \tag{F.23}$$

Substitute (F.20) and (F.23) into (F.22) and obtain

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - [2\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}_t)]^{-1} 2\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} [\boldsymbol{f}(\boldsymbol{x}_t) - \boldsymbol{y}] \\ &= \boldsymbol{x}_t + [\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}_t)]^{-1} \nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} [\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t)], \end{aligned}$$

which is identical to (F.18). The formula (F.18) embodies essentially the same spirit as the Newton method (F.22) but differs from (F.22) only in how the Hessian matrix $\nabla^2 q(\boldsymbol{x}_t)$ is computed: (F.22) relies on literal computation of the Hessian matrix $\nabla^2 q(\boldsymbol{x}_t)$, whereas (F.18) relies on approximation of the Hessian matrix $\nabla^2 q(\boldsymbol{x}_t)$ via (F.23).

In practical applications, the quasi-Newton method (F.18) and the original Newton method (F.22) tend to have similar performance, yet (F.18) tends to be computationally much more efficient than (F.22), because approximation of the Hessian matrix $\nabla^2 q(\boldsymbol{x}_t)$ via (F.23) is normally computationally much more efficient than literal computation of the Hessian matrix $\nabla^2 q(\boldsymbol{x}_t)$, especially noting that relevant computations are usually done in numerical way.

The quasi-Newton method (F.18), like the original Newton method (F.22), may enjoy fast convergence but may also encounter the problem of unstable convergence or severely oscillatory convergence. In contrast, the well-known *steepest descent method* enjoys stable convergence but suffers from comparatively slow convergence rate. To merge the (quasi) Newton method and the steepest descent method for sake of somehow benefiting from merits of both methods, *Levenberg* [29] proposes an improvement of (F.18) as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + [\nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} \nabla \boldsymbol{f}(\boldsymbol{x}_t) + \lambda \ \boldsymbol{I}]^{-1} \nabla \boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}} [\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}_t)]. \tag{F.24}$$

If the damping ratio $\lambda = 0$, then the formalism of (F.24) is reduced exactly to that of the quasi-Newton method (F.18). If the damping ratio $\lambda \to \infty$, then (F.24) becomes essentially the same to the steepest descent method. The damping ratio λ is tuned to balance effects of the quasi-Newton method and the steepest descent method.

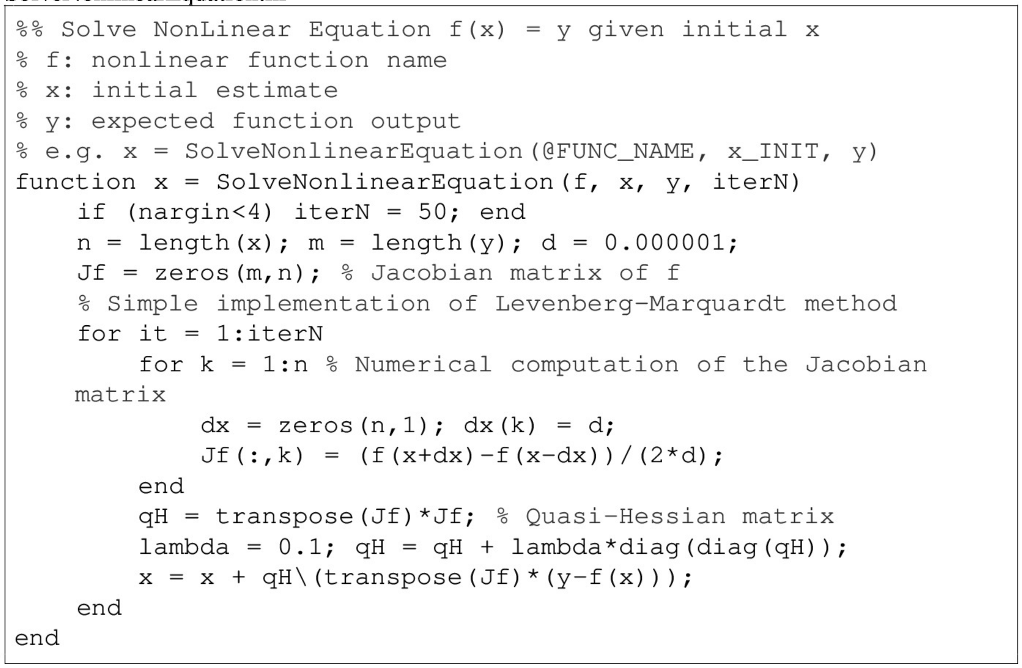
Following the spirit of the *Levenberg method* (F.24), *Marquardt* [30] modifies (F.24) by replacing the identity matrix with the diagonal matrix created from the approximated Hessian matrix, namely

$$\boldsymbol{x}_{t+1}=\boldsymbol{x}_t+\left[\nabla\boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}}\nabla\boldsymbol{f}(\boldsymbol{x}_t)+\lambda\,\mathbf{diag}(\nabla\boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}}\nabla\boldsymbol{f}(\boldsymbol{x}_t))\right]^{-1}\nabla\boldsymbol{f}(\boldsymbol{x}_t)^{\mathrm{T}}[\boldsymbol{y}-\boldsymbol{f}(\boldsymbol{x}_t)]. \tag{F.25}$$

(F.24) and (F.25) are together called the *Levenberg-Marquardt method*.

Matlab code for a simple implementation of the Levenberg-Marquardt method is given as follows. Here, the damping ratio λ is simply fixed for sake of code conciseness so that readers can easily capture the basic structure of iterative optimization. Despite its simplicity, the implementation is itself effective enough for many practical applications, at least for demonstrated practices throughout this book. Yet it is worth noting that there exist more sophisticated implementations of the Levenberg-Marquardt method [31], which usually incorporate certain adaptive mechanisms especially those for adaptive tuning of the damping ratio λ.

SolveNonlinearEquation.m



```
%% Solve NonLinear Equation f(x) = y given initial x
% f: nonlinear function name
% x: initial estimate
% y: expected function output
% e.g. x = SolveNonlinearEquation(@FUNC_NAME, x_INIT, y)
function x = SolveNonlinearEquation(f, x, y, iterN)
    if (nargin<4) iterN = 50; end
    n = length(x); m = length(y); d = 0.000001;
    Jf = zeros(m,n); % Jacobian matrix of f
    % Simple implementation of Levenberg-Marquardt method
    for it = 1:iterN
        for k = 1:n % Numerical computation of the Jacobian
    matrix
            dx = zeros(n,1); dx(k) = d;
            Jf(:,k) = (f(x+dx)-f(x-dx))/(2*d);
        end
        qH = transpose(Jf)*Jf; % Quasi-Hessian matrix
        lambda = 0.1; qH = qH + lambda*diag(diag(qH));
        x = x + qH\(transpose(Jf)*(y-f(x)));
    end
end
```

F.3.3 Parametric Nonlinear Equation

In practical applications, especially in the context of *model fitting*, one usually needs to deal with a parametric nonlinear function

$$\boldsymbol{y}_{\mathrm{O}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I}}) \tag{F.26}$$

that represents the target model of known structure $\boldsymbol{f}(\boldsymbol{x}, \cdot)$ but of unknown (to be identified) parameter set $\boldsymbol{x}$. In (F.26), $\boldsymbol{x}_{\mathrm{I}}$ and $\boldsymbol{y}_{\mathrm{O}}$ denote the actual input and output of the nonlinear function,[20] respectively.

Besides, one usually can know *a priori* pairs of the input and output of the parametric nonlinear function, denoted as

$$\{\boldsymbol{x}_{\mathrm{I},1}, \boldsymbol{y}_{\mathrm{O},1}\}, \quad \{\boldsymbol{x}_{\mathrm{I},2}, \boldsymbol{y}_{\mathrm{O},2}\}, \quad \cdots \quad , \quad \{\boldsymbol{x}_{\mathrm{I},N-1}, \boldsymbol{y}_{\mathrm{O},N-1}\}, \quad \{\boldsymbol{x}_{\mathrm{I},N}, \boldsymbol{y}_{\mathrm{O},N}\}.$$

Then the parameter set $\boldsymbol{x}$ is revealed by solving the nonlinear equation group

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},1}) &= \boldsymbol{y}_{\mathrm{O},1},\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},2}) &= \boldsymbol{y}_{\mathrm{O},2},\\ \vdots \quad & \quad \vdots\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},N-1}) &= \boldsymbol{y}_{\mathrm{O},N-1},\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},N}) &= \boldsymbol{y}_{\mathrm{O},N}.\end{aligned} \tag{F.27}$$

Group the N terms on the left side of (F.27) compactly as a large vector function and group the N terms on the right side of (F.27) compactly as a large vector, i.e.

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{f}}) \equiv \begin{matrix}\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},1})\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},2})\\ \vdots\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},N-1})\\ \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{I},N})\end{matrix}, \quad \boldsymbol{y}_{\boldsymbol{f}} \equiv \begin{matrix}\boldsymbol{y}_{\mathrm{O},1}\\ \boldsymbol{y}_{\mathrm{O},2}\\ \vdots\\ \boldsymbol{y}_{\mathrm{O},N-1}\\ \boldsymbol{y}_{\mathrm{O},N}\end{matrix},$$

where

$$\boldsymbol{x}_{\boldsymbol{f}} \equiv [\boldsymbol{x}_{\mathrm{I},1} \quad \boldsymbol{x}_{\mathrm{I},2} \quad \cdots \boldsymbol{x}_{\mathrm{I},N-1} \quad \boldsymbol{x}_{\mathrm{I},N}].$$

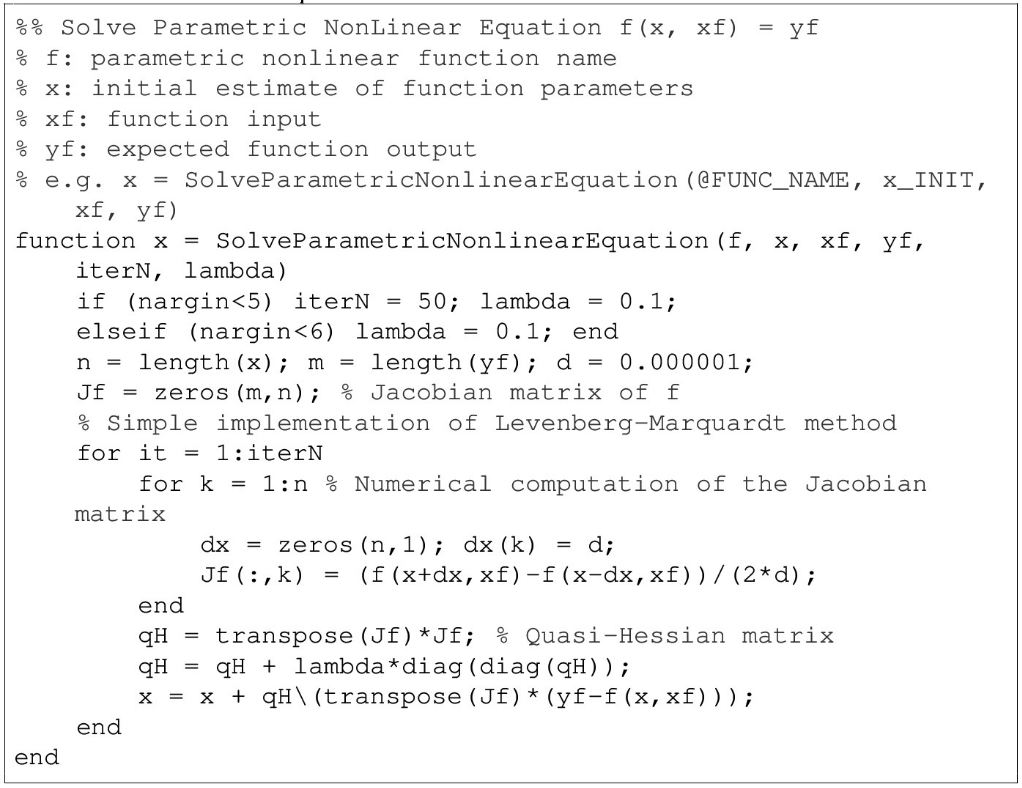
Then the nonlinear equation group (F.27), which is in terms of the parameter set $\boldsymbol{x}$, is formalized equivalently in the compact form

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{f}}) = \boldsymbol{y}_{\boldsymbol{f}}. \tag{F.28}$$

Solve (F.28) like how to solve (F.15).

Matlab code for solving the parameter set $\boldsymbol{x}$ of the nonlinear equation group (F.28) is given as follows. In fact, the Matlab code **SolveParametricNonlinearEquation.m** is essentially the same to and is adapted directly from the Matlab code **SolveNonlinearEquation.m**.

SolveParametricNonlinearEquation.m



```
%% Solve Parametric NonLinear Equation f(x, xf) = yf
% f: parametric nonlinear function name
% x: initial estimate of function parameters
% xf: function input
% yf: expected function output
% e.g. x = SolveParametricNonlinearEquation(@FUNC_NAME, x_INIT,
    xf, yf)
function x = SolveParametricNonlinearEquation(f, x, xf, yf,
    iterN, lambda)
    if (nargin<5) iterN = 50; lambda = 0.1;
    elseif (nargin<6) lambda = 0.1; end
    n = length(x); m = length(yf); d = 0.000001;
    Jf = zeros(m,n); % Jacobian matrix of f
    % Simple implementation of Levenberg-Marquardt method
    for it = 1:iterN
        for k = 1:n % Numerical computation of the Jacobian
    matrix
            dx = zeros(n,1); dx(k) = d;
            Jf(:,k) = (f(x+dx,xf)-f(x-dx,xf))/(2*d);
        end
        qH = transpose(Jf)*Jf; % Quasi-Hessian matrix
        qH = qH + lambda*diag(diag(qH));
        x = x + qH\(transpose(Jf)*(yf-f(x,xf)));
    end
end
```

Footnotes

[15] Do not take it for granted that a group of equations would have infinite solutions when the number of unknown variables is more than the number of equations. For example, the equation group

$$\begin{cases} x_1 + 2x_2 + 3x_3 & = 6 \\ 2x_1 + 4x_2 + 6x_3 & = 11 \end{cases}$$

has no solution at all, though it has more unknown variables than the equations.

[16] Explicit m-script implementation of the *SolveLinearEquation* function is given, simply to facilitate readers' understanding of how the generic linear equation (F.1) is solved. In fact, the Matlab built-in operation $x = A \setminus y$ can be used directly to perform the same function.

[17] Readers may doubt whether negligence of the scaling factor in homogeneous linear equation solutions is valid. In fact, in practical applications, whenever a homogeneous linear equation is established according to certain constraints, the scaling factor tends to play no role in the constraints. Otherwise, the scaling factor should be expressed explicitly somehow in the equations, which obviously contradicts absence of the scaling factor in the homogeneous linear equation. Take homography as example, the homography matrix, which may be calibrated by solving a homogeneous linear equation in terms of its elements, has no influence at all on the plane-to-plane bijective relationship characterized by it if it changes only by a scaling factor.

[18] As explained in Sect. 6.2.1 in Chapt. 6, the least mean squares method and the least squares method are not distinguished from each other throughout this book.

[19] Readers may wonder how “close” it is or what the author means by “close” exactly? The author has to admit ambiguity of the statement and the sense of “close” is rather qualitative than quantitative. In fact, readers may roughly understand “close” in the following way: the difference between $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{y}$ may be somewhat negligible in certain computations.

[20] The term “nonlinear” by no means implies that the involved function is always a nonlinear function and can never be a linear function. Here, it just implies that the involved function is generic or arbitrary instead of being limited to cases where linear modelling should and could be adopted.

Appendix G

Artificial Neural Networks

Since more than half a century ago, researchers have been studying the possibility of using connections of neuron-like units to realize desired input-output mapping [32, 33]. A system of such connected neuron-like units is called an **artificial neural network** or **ANN** for short.[21]

The working mechanism of an artificial neural network is rather simple locally: each unit takes a combination (normally linear combination) of output from a number of other units as input and then generates its output from the combined input usually via nonlinear mapping. For example, given a neuron-like unit (the one with "*b*") illustrated in Figure G.1, the unit input x_i is formed by a linear combination of output from four other units y_{i_1}, y_{i_2}, y_{i_3}, and y_{i_4} as

$$x_\mathrm{i} = w_{\mathrm{i}_1} y_{\mathrm{i}_1} + w_{\mathrm{i}_2} y_{\mathrm{i}_2} + w_{\mathrm{i}_3} y_{\mathrm{i}_3} + w_{\mathrm{i}_4} y_{\mathrm{i}_4} + b. \tag{G.1}$$

One can generalize (G.1) by replacing the subscript "4" with a generic index "*N*"

$$x_\mathrm{i} = w_{\mathrm{i}_1} y_{\mathrm{i}_1} + w_{\mathrm{i}_2} y_{\mathrm{i}_2} + \cdots + w_{\mathrm{i}_{N-1}} y_{\mathrm{i}_{N-1}} + w_{\mathrm{i}_N} y_{\mathrm{i}_N} = \sum_{k=1}^{\mathrm{N}} w_{\mathrm{i}_k} y_{\mathrm{i}_k} + b. \tag{G.2}$$

The neuron-like unit generates its output y_i from the combined input x_i via nonlinear mapping. The simplest example of such nonlinear mapping is the thresholding function

$$y_\mathrm{i} = \begin{cases} 1, & x_\mathrm{i} > 0 \\ 0, & x_\mathrm{i} \leqslant 0 \end{cases} \tag{G.3}$$

The *logistic activation function* [34, 35] also known as the *logistic sigmoid function*

$$y_\mathrm{i} = \tfrac{1}{1+\mathrm{e}^{-x_\mathrm{i}}} \tag{G.4}$$

instead of (G.3) is much more commonly adopted in practical applications, because (G.4) can still provide the function of thresholding on the whole and it is smooth for differentiation as well. Besides the logistic function that characterizes saturating nonlinearity, the *rectified linear unit* or *ReLU* function

$$y_\mathrm{i} = \max\{0, x_\mathrm{i}\} \tag{G.5}$$

that characterizes non-saturating nonlinearity tends to be preferred nowadays.

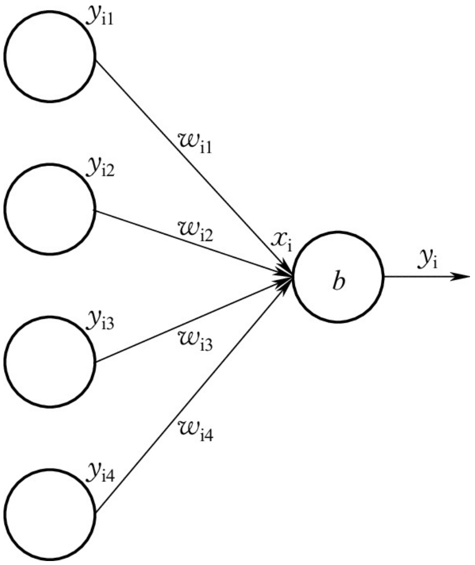


Fig. G.1 Neuron-like unit

G.1 Power of Hidden Units for Arbitrary Mapping

The simplest kind of artificial neural network is an ANN of only two layers of units, i.e. the input layer of units and the output layer of units. The support vector machine can be regarded as a typical example of such purely input-output two-layer ANN. The biggest shortcoming of the input-output two-layer ANN is that its application is limited to cases with linearly separable features .[22] A well-known example that highlights such shortcoming is the *XOR* (*eXclusive OR*) problem which can be treated equivalently as assigning four feature points (0, 0), (0, 1), (1, 0), and (1, 1) to two groups: (0, 0) and (1, 1) to one group, whereas (0, 1) and (1, 0) to the other group.

It is pointed out in [33] that an artificial neural network, with one more layer of *hidden units*, i.e. *internal representation* besides the input layer of units and the output layer of units, will possess the ability to perform any mapping from input to output, if the layer of hidden units is large enough. A not vigorous yet intuitively comprehensible explanation for the “omnipotent” mapping ability brought by hidden units: each hidden unit somehow represents a hyperplane based bisection mapping for feature space. Geometrically speaking, one can always use a large enough number of hyperplanes to segment an arbitrary set of feature points out in feature space, which implies that one can also use a large enough number of hidden units to approach any mapping.

Take the point separation problem illustrated in Fig. G.2 as example. The task is to perform a 2-dimensions-to-2-dimensions mapping

$$\boldsymbol{f} \ : \ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2 \rightarrow \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \in \mathbb{R}^2, \tag{G.6}$$

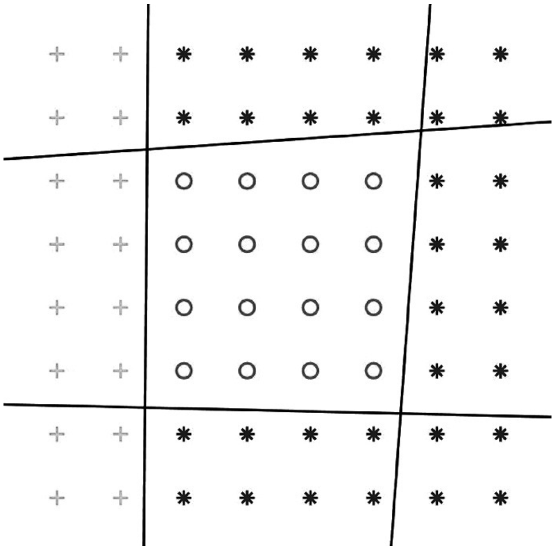


Fig. G.2 Point separation in two-dimensional space

following the logic below

$$y_1 = \begin{cases} 1, & [x_1 \quad x_2]^{\mathrm{T}} \quad \text{belongs to circle ``o'' points} \\ 0, & \text{otherwise} \end{cases}$$

$$y_2 = \begin{cases} 1, & [x_1 \quad x_2]^{\mathrm{T}} \quad \text{belongs to cross ``+'' points} \\ 0, & \text{otherwise} \end{cases}$$

In other words, there are three states of (y_1, y_2) namely (1, 0), (0, 1), (0, 0) which correspond to the category of circle “o” points, the category of cross “+” points, and the category of star “*” points respectively.

Geometric intuition tells that the point separation problem can be solved with the help of four lines (“hyperplanes” for two-dimensional space) as indicated by the two roughly vertical lines and two roughly horizontal lines in Fig. G.2. Each of the four lines dictates a bisection mapping, the $(x_1, x_2) \rightarrow y_1$ mapping can be achieved by an intersection of all the four bisection mapping results, whereas the $(x_1, x_2) \rightarrow y_2$ mapping can be achieved simply by the bisection mapping result associated with the left vertical line. Therefore, the intended mapping (G.6) can be achieved by linear combination of the four bisection mapping results.

The illustrated point separation problem itself is easy and is out of concern. What the author intends to focus in handling of this example is to demonstrate next how an artificial neural networks can successfully perform mapping of linearly inseparable feature points and can automatically learn some results that are consistent with one’s geometric intuition.

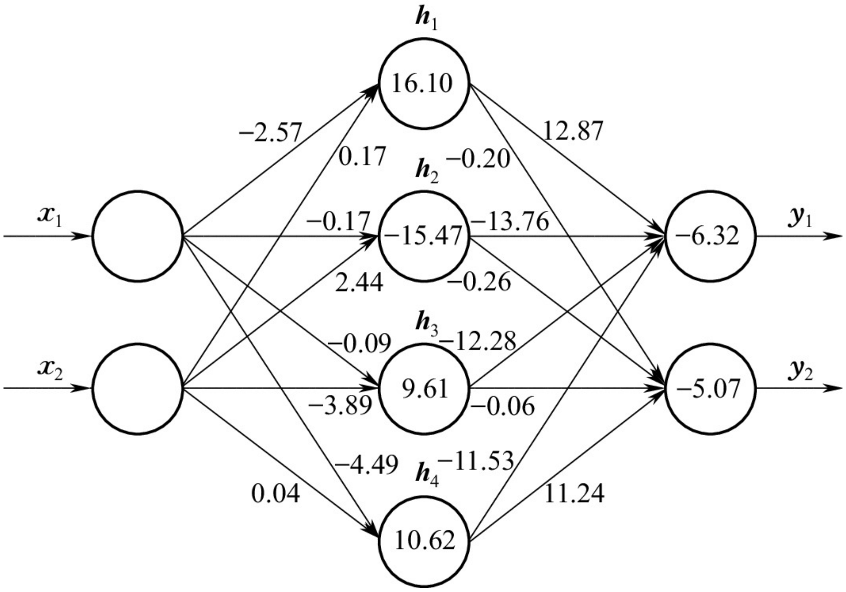

h_1
16.10
−2.57
0.17
12.87
h_2
−0.20
x_1
−0.17
−15.47
−13.76
y_1
−6.32
2.44
−0.26
h_3
−0.09
−12.28
x_2
9.61
y_2
−5.07
−3.89
−0.06
−4.49
h_4
−11.53
0.04
11.24
10.62

Fig. G.3 Three-layers (one-hidden layer) ANN for point separation

As explained above, each hidden unit somehow represents a hyperplane based bisection mapping for feature space, and four lines which dictate four bisection mappings are sufficient to handle the concrete task of point separation. So the author designs an artificial neural network in which a layer of four hidden units is set, as illustrated in Fig. G.3. Train it with the labelled feature points. Matlab code for artificial neural network mapping and learning is given as follows.

ANNBatch.m

```
%% Artificial Neural Network (3-layers) batch computation
% annV = [w_(1,1); ...; w_(1,n); w_(1,n+1); ...;
%         w_(h,1); ...; w_(h,n); w_(h,n+1);
%         w2_(1,1); ...; w2_(1,h); w2_(1,h+1)]; ...
%         w2_(m,1); ...; w2_(m,h); w2_(m,h+1)];
% n : Input dimension (number of input units)
% h : Hidden dimension (number of hidden units)
% m : Output dimension (number of output units)
% c : Number of cases (input-output pairs)
% xB: ANN input batch (n-by-c)
% yB: ANN output batch (m*c-by-1)
function yB = ANNBatch(annV, aI)
    xB = aI.xB; n = aI.n; h = aI.h; m = aI.m; c = size(xB,2);
    W = reshape(annV(1:(n+1)*h),n+1,h)’;
    W2 = reshape(annV((n+1)*h+1:end),h+1,m)’;
    hBi = W*[xB;ones(1,c)]; % Input batch of hidden units
    hBo = 1./(1+exp(-hBi)); % Output batch of hidden units
    yB = 1./(1+exp(-reshape(W2*[hBo;ones(1,c)],[],1)));
    yB = [yB; aI.wcost*annV];
end
```

ANNLearnNaive.m

```
%% ANN learning via direct (naive) optimization
function annV = ANNLearnNaive(aI, iterN)
    annV = rand((aI.n+1)*aI.h+(aI.h+1)*aI.m,1)-0.5;
    fprintf(’Initial ANN residual: %f\n’, ANNResidual(annV, aI))
    ;
    annV = SolveParametricNonlinearEquation(@ANNBatch, ...
        annV, aI, [aI.yE;zeros(size(annV))], iterN, 1.0);
    fprintf(’Final ANN residual: %f\n’, ANNResidual(annV, aI));

    function res = ANNResidual(annV, aI)
        yB = ANNBatch(annV, aI);
        res = sum((yB-[aI.yE;zeros(size(annV))]).^2);
    end
end
```

The complete demonstration code of performing point separation via artificial neural network is given as follows (see the third part of the Matlab script **demoANN.m**). The complete demonstration code of performing XOR operation and parity checking by two artificial neural networks respectively is also contained in the Matlab script **demoANN.m** (see the first part and the second part respectively).

demoANN.m

```
%% Demo arbitrary mapping ability of ANN
format compact
fprintf(’\nDemo xor via ANN => \n’);
clear; aI.xB = [rem(0:3,2); rem(floor((0:3)/2),2)];
aI.n = 2; aI.h = 2; aI.m = 1; aI.yE = rem(sum(aI.xB,1),2)’;
aI.wcost = 0.01; annV = ANNLearnNaive(aI,5000);
yB = ANNBatch(annV, aI); yB = yB(1:end-length(annV))’;
yB2 = ANNThresholding(yB); fprintf(’annV : ’); disp(annV’);
fprintf(’ANN performance :\n’); disp([aI.xB; yB; yB2]);

fprintf(’\nDemo parity checking via ANN => \n’);
clear; aI.xB = [rem(0:7,2);rem(floor((0:7)/2),2);rem(floor((0:7)
    /4),2)];
aI.n = 3; aI.h = 3; aI.m = 1; aI.yE = rem(sum(aI.xB,1),2)’;
aI.wcost = 0.01; annV = ANNLearnNaive(aI,5000);
yB = ANNBatch(annV, aI); yB = yB(1:end-length(annV))’;
yB2 = ANNThresholding(yB); fprintf(’annV : ’); disp(annV’);
fprintf(’ANN performance :\n’); disp([aI.xB; yB; yB2]);

fprintf(’\nDemo point separation via ANN => \n’);
clear; x1 = repmat(1:8,8,1); x2 = repmat((1:8)’,1,8);
y1=zeros(8,8); y1(3:6,3:6)=1; y2=zeros(8,8); y2(:,1:2)=1;
aI.xB = [reshape(x1,1,[]); reshape(x2,1,[])];
aI.n = 2; aI.h = 4; aI.m = 2;
aI.yE = reshape([reshape(y1,1,[]);reshape(y2,1,[])],[],1);
aI.wcost = 0.01; annV = ANNLearnNaive(aI,5000);
yB = ANNBatch(annV, aI); yB = yB(1:end-length(annV))’;
yB1 = yB(1:2:end); yB1 = reshape(yB1,8,8);
yB2 = yB(2:2:end); yB2 = reshape(yB2,8,8);
yT1 = ANNThresholding(yB1); yT2 = ANNThresholding(yB2);
fprintf(’annV : ’); disp(annV’);
fprintf(’ANN performance :\n’); yB1, yB2, yT1, yT2
ia = find(yT1==1); ib = find(yT2==1); ic = find(yT1+yT2==0);
```

ANNThresholding.m

```
function yT = ANNThresholding(y)
    i0 = find(y<0.2); i1 = find(y>0.8);
    yT = zeros(size(y))+0.5; yT(i0) = 0; yT(i1) = 1;
end
```

The provided Matlab script **ANNLearnNaive.m**, which realizes artificial neural network learning in a naive way, is purely for demonstration purpose. Why naive will be clarified latter. Despite its naiveness, the Matlab script **ANNLearnNaive.m** can satisfy the requirement of training small artificial neural networks such as those in examples demonstrated here. The involved Matlab code **SolveParametricNonlinearEquation.m** is presented in Section F.3 in Appendix F.

Readers can try the provided Matlab code and will find that proper parameters of the artificial neural network are learned automatically. For example, the connection weights and unit biases demonstrated in Figure G.3 are learned automatically.[23] The four lines illustrated in Fig. G.2, which dictate four bisection mappings to handle the point separation problem, correspond right to the automatically learned parameters between the input units and the hidden units illustrated in Fig. G.3

$$\begin{aligned}
L_{h_1}: &\quad -2.57x_1 + 0.17x_2 + 16.10 = 0,\\
L_{h_2}: &\quad -0.17x_1 + 2.44x_2 - 15.47 = 0,\\
L_{h_3}: &\quad -0.09x_1 - 3.89x_2 + 9.61 = 0,\\
L_{h_4}: &\quad -4.49x_1 + 0.04x_2 + 10.62 = 0.
\end{aligned}$$

G.2 Back-Propagation for Tractable ANN Optimization

Thanks to the power brought by hidden units, artificial neural networks enjoy the “omnipotent” ability of learning an arbitrary mapping from input to output. Examples together with Matlab demonstration code have been presented to facilitate readers’ understanding of such charm of artificial neural networks.

G.2.1 Note on Naive Implementation

Now revisit the provided Matlab script **ANNLearnNaive.m** that realizes artificial neural network learning for demonstrated examples (including the examples of performing XOR operation and parity checking as well). The code architecture of the Matlab script **ANNLearnNaive.m** is simple and the code logic is direct and easily comprehensible. However, instead of “praising” the implementation given in the Matlab script **ANNLearnNaive.m**, the author intends to point out its naiveness here, because an understanding of the naiveness of such plausibly effective implementation enables a better appreciation of the famous back-propagation method [34, 35] which will be described soon.

In the given implementation, the artificial neural network mapping is treated holistically as a parametric nonlinear function, and its parameters are learned via direct optimization in the generic framework of solving parametric nonlinear equations numerically. This generic framework, which is presented in Sect. F.3 in Appendix F, relies on a large amount of repetitive computations using the involved parametric nonlinear function. More specifically, in each iteration of numeric optimization, partial derivatives of the parametric nonlinear function are approximated by differencing function output values that are obtained by a large amount of repetitive invocation of the parametric nonlinear function given different optimization variable input values, as shown in the following partial code excerpt of the Matlab script **SolveParametricNonlinearEquation.m**.

```
% Partial code excerpt of SolveParametricNonlinearEquation.m
for k = 1:n % Numerical computation of the Jacobian matrix
    dx = zeros(n,1); dx(k) = d;
    Jf(:,k) = (f(x+dx,xf)-f(x-dx,xf))/(2*d);
end
```

Such repetitive invocation of the parametric nonlinear function can be acceptable for practical applications involving small optimization problems, but is computational forbidding for practical applications involving large optimization problems. Neural networks especially deep neural networks based applications usually involve large optimization problems. In fact, a single invocation of the artificial neural network mapping (treated holistically as a function) may take considerable computations and hence is not computationally cheap, not to say a large amount of repetitive invocation of the artificial neural network mapping. The naiveness of the implementation given in the Matlab script **ANNLearnNaive.m** consists right in invocation of the artificial neural network mapping as a holistic function for computation of partial derivatives in involved numeric optimization.

G.2.2 Compute Partial Derivatives via Back-Propagation

Computation of partial derivatives of a parametric function is indispensable to numeric optimization of the parametric function when analytic methods are adopted (in contrast with combinatorial optimization style methods such as genetic algorithms and simulated annealing techniques). A natural idea for computation of partial derivatives is to treat the artificial neural network mapping as a *black box*, i.e. treat it holistically as a parametric nonlinear function and approximate partial derivatives by differencing function output values from the function outside perspective. This is the idea adopted by the implementation given in the Matlab script **ANNLearnNaive.m**, but its naiveness has just been pointed out previously.

Another idea for computation of partial derivatives is to treat the artificial neural network mapping as a *white box*, i.e. treat it as a system of accessible composing sub-functions and compute partial derivatives by taking advantage of its internal structures from the system inside perspective. But the question is how to put the idea into practice?

The milestone articles [34, 35] report a method coined as **back-propagation** that puts the idea into practice successfully and enables tractable optimization (learning) of generic artificial neural networks. Thanks to the back-propagation method and current hardware support, an extremely wide range of practical applications can benefit from deep neural networks nowadays.

The spirit of the back-propagation method consists in computing partial derivatives of artificial neural network parameters gradually from the output layer of units back towards the input layer of units. More specifically, take the three-layers (one-hidden layer) artificial neural network as a paradigm for derivation of back-propagation formalisms, which can be extended to generic artificial neural networks without difficulty.

Suppose there are *n* input units x_1, x_2, ..., x_n; there are *t* hidden units $\{h_1^{\mathrm{I}}, h_1^{\mathrm{O}}\}$, $\{h_2^{\mathrm{I}}, h_2^{\mathrm{O}}\}$, ..., $\{h_t^{\mathrm{I}}, h_t^{\mathrm{O}}\}$ where h_i^{I} and h_i^{O} denote generically the input and output of the *i*-th hidden unit respectively; there are *m* output units $\{y_1^{\mathrm{I}}, y_1^{\mathrm{O}}\}$, $\{y_2^{\mathrm{I}}, y_2^{\mathrm{O}}\}$, ..., $\{y_m^{\mathrm{I}}, y_m^{\mathrm{O}}\}$ where y_i^{I} and y_i^{O} denote generically the input and output of the *i*-th output unit respectively. Suppose the expected output values of the *m* output units are y_1,y_2, ..., y_m respectively. Further denote the weight connecting x_j to h_i^{I} generically as $w_{i,j}^{\mathrm{h}}$ and the bias at the *i*-th hidden unit as $w_{i,n+1}^{\mathrm{h}}$ where $j \in \{1, \cdots, n\}$ and $i \in \{1, \cdots, t\}$; denote the weight connecting h_j^{O} to y_i^{I} generically as $w_{i,j}^{\mathrm{y}}$ and the bias at the *i*-th output unit as $w_{i,t+1}^{\mathrm{y}}$ where $j \in \{1, \cdots, t\}$ and $i \in \{1, \cdots, m\}$. Denote compactly

$$\boldsymbol{x} \equiv \begin{matrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{matrix}, \quad \boldsymbol{h}^{\mathrm{I}} \equiv \begin{matrix} h_1^{\mathrm{I}} \\ h_2^{\mathrm{I}} \\ \vdots \\ h_t^{\mathrm{I}} \end{matrix}, \quad \boldsymbol{h}^{\mathrm{O}} \equiv \begin{matrix} h_1^{\mathrm{O}} \\ h_2^{\mathrm{O}} \\ \vdots \\ h_t^{\mathrm{O}} \end{matrix}, \quad \boldsymbol{y}^{\mathrm{I}} \equiv \begin{matrix} y_1^{\mathrm{I}} \\ y_2^{\mathrm{I}} \\ \vdots \\ y_m^{\mathrm{I}} \end{matrix}, \quad \boldsymbol{y}^{\mathrm{O}} \equiv \begin{matrix} y_1^{\mathrm{O}} \\ y_2^{\mathrm{O}} \\ \vdots \\ y_m^{\mathrm{O}} \end{matrix}, \quad \boldsymbol{y} \equiv \begin{matrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{matrix},$$

$$\boldsymbol{w}^{\mathrm{h}} \equiv \begin{matrix} w_{1,1}^{\mathrm{h}} & w_{1,2}^{\mathrm{h}} & \cdots & w_{1,n}^{\mathrm{h}} & w_{1,n+1}^{\mathrm{h}} \\ w_{2,1}^{\mathrm{h}} & w_{2,2}^{\mathrm{h}} & \cdots & w_{2,n}^{\mathrm{h}} & w_{2,n+1}^{\mathrm{h}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{t,1}^{\mathrm{h}} & w_{t,2}^{\mathrm{h}} & \cdots & w_{t,n}^{\mathrm{h}} & w_{t,n+1}^{\mathrm{h}} \end{matrix},$$

$$\boldsymbol{w}^{\mathrm{y}} \equiv \begin{matrix} w_{1,1}^{\mathrm{y}} & w_{1,2}^{\mathrm{y}} & \cdots & w_{1,t}^{\mathrm{y}} & w_{1,t+1}^{\mathrm{y}} \\ w_{2,1}^{\mathrm{y}} & w_{2,2}^{\mathrm{y}} & \cdots & w_{2,t}^{\mathrm{y}} & w_{2,t+1}^{\mathrm{y}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{m,1}^{\mathrm{y}} & w_{m,2}^{\mathrm{y}} & \cdots & w_{m,t}^{\mathrm{y}} & w_{m,t+1}^{\mathrm{y}} \end{matrix}.$$

Adopt an error function or loss function of the least mean squares form

$$E(\boldsymbol{w}^{\mathrm{h}}, \boldsymbol{w}^{\mathrm{y}}) = \tfrac{1}{2} \textstyle\sum_c \left\|\boldsymbol{y}^{\mathrm{O}} - \boldsymbol{y}\right\|_2^2 = \tfrac{1}{2} \sum_c \sum_{i=1}^{m} \left(y_i^{\mathrm{O}} - y_i\right)^2 \tag{G.7}$$

where *c* denotes the case index or the input-output pair index, but the author intentionally omits explicit expression of it for formalism conciseness. So (G.7) is actually

$$E(\boldsymbol{w}^{\mathrm{h}}, \boldsymbol{w}^{\mathrm{y}}) = \frac{1}{2}\sum_c \|\boldsymbol{y}_c^{\mathrm{O}} - \boldsymbol{y}_c\|_2^2 = \frac{1}{2}\sum_c \sum_{i=1}^m \left(y_{i,c}^{\mathrm{O}} - y_{i,c}\right)^2$$

and readers had better bear in mind there is an implicit subscript c for elements of $\boldsymbol{x}$, $\boldsymbol{h}^{\mathrm{I}}$, $\boldsymbol{h}^{\mathrm{O}}$, $\boldsymbol{y}^{\mathrm{I}}$, $\boldsymbol{y}^{\mathrm{O}}$, and $\boldsymbol{y}$ in following derivation of formalisms.

Compute partial derivatives at output units (including weights $\boldsymbol{w}^{\mathrm{y}}$)

First, compute partial derivatives at $\boldsymbol{y}^{\mathrm{O}}$ and back-propagate the obtained results

$$\frac{\partial E}{\partial y_i^{\mathrm{O}}} = y_i^{\mathrm{O}} - y_i, \quad i \in \{1, \cdots, m\}. \tag{G.8}$$

Second, compute partial derivatives at $\boldsymbol{y}^{\mathrm{I}}$ using the *chain rule* and back-propagate the obtained results[24]

$$\frac{\partial E}{\partial y_i^{\mathrm{I}}} = \frac{\partial E}{\partial y_i^{\mathrm{O}}}\frac{\mathrm{d}y_i^{\mathrm{O}}}{\mathrm{d}y_i^{\mathrm{I}}} = \frac{\partial E}{\partial y_i^{\mathrm{O}}}(1 - y_i^{\mathrm{O}})y_i^{\mathrm{O}}, \quad i \in \{1, \cdots, m\}. \tag{G.9}$$

Third, compute partial derivatives at $\boldsymbol{w}^{\mathrm{y}}$ using the chain rule and back-propagate the obtained results

$$\frac{\partial E}{\partial w_{i,j}^{\mathrm{y}}} = \frac{\partial E}{\partial y_i^{\mathrm{I}}}\frac{\partial y_i^{\mathrm{I}}}{\partial w_{i,j}^{\mathrm{y}}} = \frac{\partial E}{\partial y_i^{\mathrm{I}}} h_j^{\mathrm{O}}, \tag{G.10a}$$

$$\frac{\partial E}{\partial w_{i,t+1}^{\mathrm{y}}} = \frac{\partial E}{\partial y_i^{\mathrm{I}}}\frac{\partial y_i^{\mathrm{I}}}{\partial w_{i,t+1}^{\mathrm{y}}} = \frac{\partial E}{\partial y_i^{\mathrm{I}}}, \quad j \in \{1, \cdots, t\}, i \in \{1, \cdots, m\}. \tag{G.10b}$$

Compute partial derivatives at hidden units (including weights $\boldsymbol{w}^{\mathrm{h}}$)

First, compute partial derivatives at $\boldsymbol{h}^{\mathrm{O}}$ using the chain rule and back-propagate the obtained results

$$\frac{\partial E}{\partial h_i^{\mathrm{O}}} = \sum_{j=1}^m \frac{\partial E}{\partial y_j^{\mathrm{I}}}\frac{\partial y_j^{\mathrm{I}}}{\partial h_i^{\mathrm{O}}} = \sum_{j=1}^m \frac{\partial E}{\partial y_j^{\mathrm{I}}} w_{j,i}^{\mathrm{y}}, \quad i \in \{1, \cdots, t\}. \tag{G.11}$$

Second, compute partial derivatives at $\boldsymbol{h}^{\mathrm{I}}$ using the chain rule and back-propagate the obtained results

$$\frac{\partial E}{\partial h_i^{\mathrm{I}}} = \frac{\partial E}{\partial h_i^{\mathrm{O}}}\frac{\mathrm{d}h_i^{\mathrm{O}}}{\mathrm{d}h_i^{\mathrm{I}}} = \frac{\partial E}{\partial h_i^{\mathrm{O}}}(1 - h_i^{\mathrm{O}})h_i^{\mathrm{O}}, \quad i \in \{1, \cdots, t\}. \tag{G.12}$$

Third, compute partial derivatives at $\boldsymbol{w}^{\mathrm{h}}$ using the chain rule[25]

$$\frac{\partial E}{\partial w^{\mathrm{h}}_{i,j}} = \frac{\partial E}{\partial h^{\mathrm{I}}_i}\frac{\partial h^{\mathrm{I}}_i}{\partial w^{\mathrm{h}}_{i,j}} = \frac{\partial E}{\partial h^{\mathrm{I}}_i} x_j, \tag{G.13a}$$

$$\frac{\partial E}{\partial w^{\mathrm{h}}_{i,n+1}} = \frac{\partial E}{\partial h^{\mathrm{I}}_i}\frac{\partial h^{\mathrm{I}}_i}{\partial w^{\mathrm{h}}_{i,n+1}} = \frac{\partial E}{\partial h^{\mathrm{I}}_i}, \quad j \in \{1, \cdots, n\}, i \in \{1, \cdots, t\}. \tag{G.13b}$$

Compact Formalism

One can resort to the *Hamadard product* $\odot$ and the *Hamadard division* $\oslash$, namely the element-wise product and division , and formalize (G.8), (G.9), (G.10), (G.11), (G.12), and (G.13) compactly as

$$\frac{\partial E}{\partial \boldsymbol{y}^{\mathrm{O}}} = \boldsymbol{y}^{\mathrm{O}} - \boldsymbol{y} \tag{G.14a}$$

$$\frac{\partial E}{\partial \boldsymbol{y}^{\mathrm{I}}} = \frac{\partial E}{\partial \boldsymbol{y}^{\mathrm{O}}} \odot (1 - \boldsymbol{y}^{\mathrm{O}}) \odot \boldsymbol{y}^{\mathrm{O}} \tag{G.14b}$$

$$\frac{\partial E}{\partial \boldsymbol{w}^{\mathrm{y}}} = \frac{\partial E}{\partial \boldsymbol{y}^{\mathrm{I}}} \left[\left(\boldsymbol{h}^{\mathrm{O}}\right)^{\mathrm{T}} \quad 1 \right] \tag{G.14c}$$

$$\frac{\partial E}{\partial \boldsymbol{h}^{\mathrm{O}}} = [\boldsymbol{I} \quad \boldsymbol{0}] \left(\boldsymbol{w}^{\mathrm{y}}\right)^{\mathrm{T}} \frac{\partial E}{\partial \boldsymbol{y}^{\mathrm{I}}} \tag{G.14d}$$

$$\frac{\partial E}{\partial \boldsymbol{h}^{\mathrm{I}}} = \frac{\partial E}{\partial \boldsymbol{h}^{\mathrm{O}}} \odot (1 - \boldsymbol{h}^{\mathrm{O}}) \odot \boldsymbol{h}^{\mathrm{O}} \tag{G.14e}$$

$$\frac{\partial E}{\partial \boldsymbol{w}^{\mathrm{h}}} = \frac{\partial E}{\partial \boldsymbol{h}^{\mathrm{I}}} \left[\boldsymbol{x}^{\mathrm{T}} \quad 1 \right] \tag{G.14f}$$

Readers can see conciseness of the formalism described in (G.14) which can even better convey regularity of the back-propagation equations. The author believes that after observing regularity conveyed by (G.14) readers would easily write out the back-propagation equations for a generic artificial neural network potentially with more layers of units.

The artificial neural network (forward) mapping can also be formalized compactly with the help of the element-wise operation notions as

$$\boldsymbol{h}^{\mathrm{I}} = \boldsymbol{w}^{\mathrm{h}} \boldsymbol{x}^{\mathrm{A}} = \boldsymbol{w}^{\mathrm{h}} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \tag{G.15a}$$

$$\boldsymbol{h}^{\mathrm{O}} \quad = 1 \oslash \left(1 + \mathrm{e}^{[-\boldsymbol{h}^{\mathrm{I}}]}\right) \tag{G.15b}$$

$$\boldsymbol{y}^{\mathrm{I}} \quad = \boldsymbol{w}^{\mathrm{y}} \left(\boldsymbol{h}^{\mathrm{O}}\right)^{\mathrm{A}} = \boldsymbol{w}^{\mathrm{y}} \begin{bmatrix} \boldsymbol{h}^{\mathrm{O}} \\ 1 \end{bmatrix} \tag{G.15c}$$

$$\boldsymbol{y}^{\mathrm{O}} \quad = 1 \oslash \left(1 + \mathrm{e}^{[-\boldsymbol{y}^{\mathrm{I}}]}\right) \tag{G.15d}$$

where $\boldsymbol{x}^{\mathrm{A}}$ and $\left(\boldsymbol{h}^{\mathrm{O}}\right)^{\mathrm{A}}$ are augmented vectors of $\boldsymbol{x}$ and $\boldsymbol{h}^{\mathrm{O}}$ respectively, and $\mathrm{e}^{[\cdot]}$ denotes the *element-wise exponential function*.

G.2.3 Incremental Learning (Refinement) of ANN Parameters

The artificial neural network parameters are learned incrementally. The parameter increments follow the recursive relationship

$$\Delta\boldsymbol{w}^{\mathrm{y}}(k) \quad = -\beta \frac{\partial.E}{\partial \boldsymbol{w}^{\mathrm{y}}} + \alpha \Delta\boldsymbol{w}^{\mathrm{y}}(k-1), \tag{G.16a}$$

$$\Delta\boldsymbol{w}^{\mathrm{h}}(k) \quad = -\beta \frac{\partial.E}{\partial \boldsymbol{w}^{\mathrm{h}}} + \alpha \Delta\boldsymbol{w}^{\mathrm{h}}(k-1), \tag{G.16b}$$

$$\boldsymbol{w}^{\mathrm{y}}(k) \quad = \boldsymbol{w}^{\mathrm{y}}(k-1) + \Delta\boldsymbol{w}^{\mathrm{y}}(k), \tag{G.16c}$$

$$\boldsymbol{w}^{\mathrm{h}}(k) \quad = \boldsymbol{w}^{\mathrm{h}}(k-1) + \Delta\boldsymbol{w}^{\mathrm{h}}(k), \tag{G.16d}$$

where β is called the *learning rate* and α is called the *exponential decay factor* [34, 35]. Matlab code for artificial neural network learning via back-propagation is given as follows.

ANNMapping.m

```
%% Artificial Neural Network (3-layers) batch mapping
%  annW : Wh and Wy
%  c : Number of cases (input-output pairs)
%  xB: ANN input batch (n-by-c)
%  yB: ANN output batch (m-by-c)
function yB = ANNMapping(annW, aI)
    c = size(aI.xB,2);
    hBi = (annW.Wh)*[aI.xB;ones(1,c)]; % Input batch of hidden
    units
    hBo = 1./(1+exp(-hBi)); % Output batch of hidden units
    yBi = (annW.Wy)*[hBo;ones(1,c)]; % Input batch of output
    units
    yB = 1./(1+exp(-yBi)); % Output batch of output units
end
```

ANNLearnBackPropagation.m

```
 %% ANN learning via back-propagation
function annW = ANNLearnBackPropagation(aI, iterN, lr, edf,
    annWi)
    % lr: Learning rate ; % edf: Exponential decay factor
    if (nargin<3) lr = 0.2; edf = 0.8; end
    if (nargin<5)
        % n, h, m : Input, hidden, output dimensions
        Wh = rand(aI.h,aI.n+1)-0.5; Wy = rand(aI.m,aI.h+1)-0.5;
    else Wh = annWi.Wh; Wy = annWi.Wy; end
    annW.Wh = Wh; annW.Wy = Wy; dWho = 0*Wh; dWyo = 0*Wy;
    fprintf(’Initial ANN residual: %f\n’, ANNResidual(annW, aI))
    ;
    xB = aI.xB; c = size(xB,2); yE = aI.yE;
    % Incremental learning
    for it = 1:iterN
        % ANN Forward mapping
        hBi = Wh*[xB;ones(1,c)]; hBo = 1./(1+exp(-hBi));
        yBi = Wy*[hBo;ones(1,c)]; yBo = 1./(1+exp(-yBi));
        % ANN Back-propagation
        dyBo = yBo-yE; dyBi = dyBo.*(1-yBo).*yBo;
        dhBo = Wy(:,1:end-1)’*dyBi; dhBi = dhBo.*(1-hBo).*hBo;
        dWh = 0*Wh; dWy = 0*Wy;
        for k=1:c
            dWy = dWy + dyBi(:,k)*[hBo(:,k)’,1];
            dWh = dWh + dhBi(:,k)*[xB(:,k)’,1];
        end
        dWy = -lr*(dWy/c)+edf*dWyo; dWyo = dWy; Wy = Wy+dWy;
        dWh = -lr*(dWh/c)+edf*dWho; dWho = dWh; Wh = Wh+dWh;
    end
    annW.Wh = Wh; annW.Wy = Wy;
    fprintf(’Final ANN residual: %f\n’, ANNResidual(annW, aI));

    function res = ANNResidual(annW, aI)
        yB = ANNMapping(annW, aI);
        res = sum(sum((yB-aI.yE).^2));
    end
end
```

The effect of a back-propagation learned ANN of two input units, four hidden units, and two output units for handling the previously mentioned point separation problem is demonstrated in Figure G.4. Figure G.4a illustrates the ground-truth for training and also illustrates the learned $\boldsymbol{w}^{\mathrm{h}}$, i.e. the parameters (including biases) from the input units to the hidden units, which are visualized by the four lines—Explanations on geometric meanings of parameters from input units to hidden units have already been given in Sect. G.1—Fig. G.4b demonstrates results of testing on points obtained by randomly perturbing the ground-truth points.

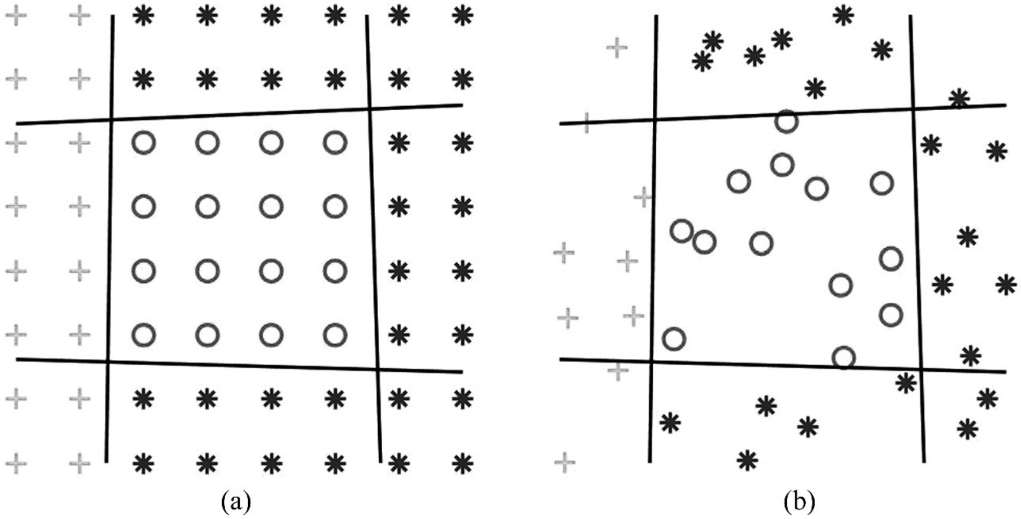

(a)
(b)

Fig. G.4 Back-propagation learned ANN of 2 input units, 4 hidden units, and 2 output units for point separation (a) training on ground-truth; (b) testing on randomly perturbed points

Consider a a new example problem more challenging than that demonstrated in FigS. G.2 and G.4. The new problem is to learn the ability of separating the two opposite parts namely the *Yin* (black, indicated by crosses) part and the *Yang* (white, indicated by circles) part in the *Tai-Ji* pattern, as illustrated in Figure G.5. The *Tai-Ji* pattern separation problem is an interesting and good example to demonstrate general non-linear mapping ability of artificial neural networks.

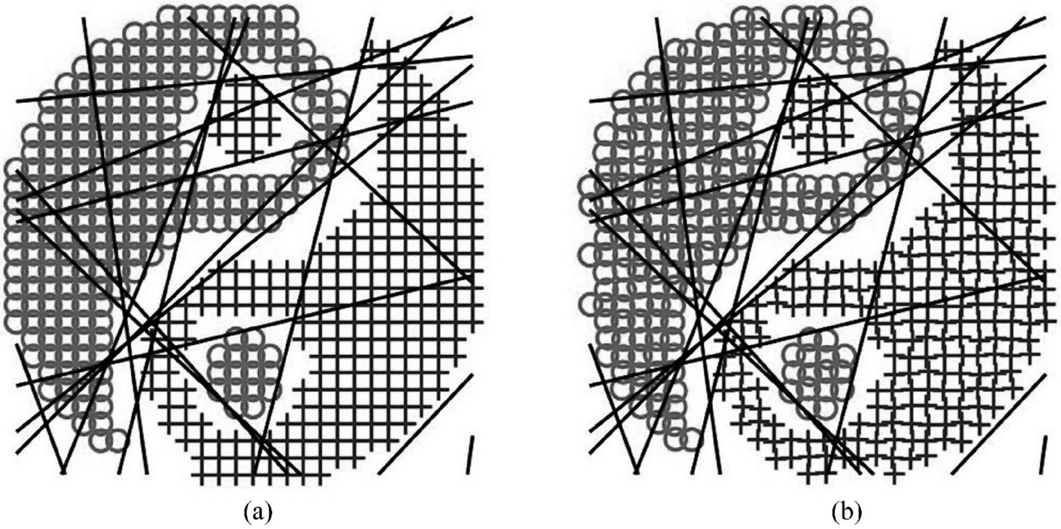

(a)
(b)

Fig. G.5 Back-propagation learned ANN of 2 input units, 16 hidden units, and 1 output units for *Tai-Ji* pattern separation (a) training on ground-truth; (b) testing on randomly perturbed points

The effect of a back-propagation learned ANN of two input units, sixteen hidden units, and one output units for *Tai-Ji* pattern separation is demonstrated in Fig. G.5. Figure G.5a illustrates the ground-truth for training and also illustrates the sixteen learned triples of $\boldsymbol{w}^{\mathrm{h}}$ parameters associated with the sixteen hidden units respectively. The sixteen learned triples of $\boldsymbol{w}^{\mathrm{h}}$ parameters are visualized by the lines in both sub-figures. Figure G.5b demonstrates results of testing on points obtained again by randomly perturbing the ground-truth points. As shown in Fig. G.5, the intended ability of *Tai-Ji* pattern separation is well achieved.

G.3 Note

The author would like to close Appendix G by providing more details behind the demonstrated example of *Tai-Ji* pattern separation, which actually reflect a number of general concerns for neural networks based applications. These general concerns include:

- *Availability of field knowledge is important*: The success of deep learning might give people an illusion that artificial neural networks especially deep neural networks are “omnipotent” for practical applications and make people neglect field knowledge (or *a priori* knowledge). Despite the power of artificial neural networks, negligence of available field knowledge is neither a good attitude nor a proper practice, because availability of field knowledge tends to be important and even indispensable to practical applications.

  For the demonstrated example of *Tai-Ji* pattern separation, readers may try the way of initializing artificial neural network parameters randomly without incorporating any *a priori* knowledge. Then readers would find that the three-layers artificial neural network, even when configured with hundreds of hidden units (and hence possessing potentially much stronger representation ability than the demonstrated one with only sixteen hidden units), would neither have desirable learning behaviour nor achieve desirable testing performance, because a large amount of artificial neural network parameters are wasted on useless regions of feature space and get stuck locally there.

  In contrast, instead of totally neglecting *a priori* knowledge, resort to a small piece of *a priori* knowledge: useful hyperplane based bisection mappings represented by hidden units should be located within the range of feature points in the *Tai-Ji* pattern and hence the $\boldsymbol{w}^{\mathrm{h}}$ parameters of hidden units had better be initialized according to randomly scattered lines (“hyperplanes” in the two-dimensional space) within the *Tai-Ji* pattern range. For the purpose, shift the coordinates system origin to the centroid of feature points in the *Tai-Ji* pattern, and initialize the three elements of each row of $\boldsymbol{w}^{\mathrm{h}}$ randomly as

  $$[\cos\theta \quad \sin\theta \quad d] \Rightarrow L_{\theta,d} \ : \cos\theta \ x + \sin\theta \ y + d = 0$$

  where the angle θ is generated randomly within $[0, 2\pi]$ and the distance *d* is generated randomly within the maximum distance range of the feature points. Simply by taking advantage of this small piece of *a priori* knowledge, the three-layers artificial neural network that consists only of a moderate amount of hidden units can perform desirably both in terms of learning and testing, as demonstrated in Fig. G.5.

- *Incorporation of training randomness is important*: Symmetrical factors have long since been known to be detrimental to artificial neural network learning [34, 35], especially noting that the commonly adopted back-propagation method is inherently of parallel processing flavour and tends to be more susceptible to unexpected influences caused by symmetrical factors. The synthesized feature points in the demonstrated example of *Tai-Ji* pattern are completely symmetrical, which is potentially harmful and may cause tricky learning results usually in the form of certain deadlock.

  A core idea to handle symmetrical factors that may undermine artificial neural network learning is to incorporate various kinds of training randomness, which can be data-level randomness or procedure-level randomness. For the demonstrated example of *Tai-Ji* pattern separation, procedure-level randomness is incorporated. More specifically, instead of training the artificial neural network with all the feature points directly from the beginning, first initiate a randomly determined pre-training process namely training the artificial neural network with two thirds of feature points selected randomly, and then continue to train the artificial neural network with all the feature points.

  Incorporation of training randomness is important not only in the sense of handling symmetrical factors but also in more general sense of enhancing training independence. Sometimes, even certain level of quasi-randomness and quasi-independence can be considerably beneficial to machine learning, such as what is explained in [36].[26]
- *Manipulation of representation capacity is important*: As readers may notice in Fig. G.5, the lines that visualize (soft) bisection mapping functions of hidden units are redundant: some lines have rather similar bisection mappings, and some lines seem to have no effect in distinguishing feature points. So it seems that the representation capacity (reflected by the amount of hidden units) of the demonstrated artificial neural network is wasteful. However, if the number of hidden units was indeed reduced to a seemingly enough number, then the artificial neural network learning process would usually fail in the form of non-convergence. The reason is that some hidden units converge to and get stuck in states which provide no useful bisection mappings in feature space on one hand, but other hidden units are not sufficient to provide all necessary bisection mappings on the other hand.

  It is like a dilemma: to guarantee a desirable learning behaviour, one normally has to configure a representation capacity more than what is seemingly needed. Besides, in many practical applications, one cannot even have an idea of what representation capacity is seemingly needed and one may conservatively configure a representation capacity much more than what is indeed needed. However, configuring an unnecessarily large representation capacity tends to incur waste of representation capacity, and also tends to incur the *over-fitting* problem.

Here, the author can only say manipulation of representation capacity is important, but unfortunately cannot provide any general rule of thumb. Manipulation of representation capacity, which includes the practice of training neural networks with redundant representation capacity first and then reducing or simplifying neural networks, is still worth researches.

Footnotes

[21] In early years, neural networks took various names such as “hypothetical nervous systems”, “connectionist systems” in researches.

[22] Even kernel techniques are involved, like for the support vector machine, yet kernel techniques do not make input-output two-layer artificial neural networks themselves be able to handle cases with linearly inseparable features, but make (if possible) features linearly separable after kernel mapping.

[23] Due to random initialization of the ANN parameters, readers may encounter different parameters after ANN learning and may have little chance of encountering failures of ANN learning as well.

[24] Note that the logistic activation function (G.4) is adopted by default.

[25] Arrive at the input layer of units and do not need to back-propagate the obtained results any more.

[26] The reference [36] is a technical note that complements previous works [37] of the team directed by the author.

References

1. S.M. Ross, *Introduction to Probability Models* (Elsevier, 2019)
2. N. Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series* (MIT Press, 1949)
3. H. Li, *Control Theory for Practical Applications* (Shanghai Jiao Tong University Press, 2024)
4. H. Li, *Fundamentals and applications of recursive estimation theory* (Shanghai Jiao Tong University Press, 2022)
5. R. Deriche, Recursively implementing the gaussian and its derivatives. *INRIA Research Report*, RR-1893 (1993)
6. R. Horaud, O. Monga. *Vision Par Ordinateur: Outils Fondamentaux*. (Editions Hermès, 1995)
7. L. Vliet, I. Young, P. Verbeek. Recursive gaussian derivative filters, in *IEEE International Conference on Pattern Recognition*, pp. 509–514 (1998)
8. T. Huang, G. Yang, G. Tang, A fast two-dimensional median filtering algorithm. IEEE Trans. Acoustics, Speech, and Signal Proc. **27**(1), 13–18 (1979)
9. R. Sedgewick. *Algorithms* (Addison-Wesley Publishing Company, 1983)
10. D. Brownrigg, The weighted median filter. Commun. ACM **27**(8), 807–818 (1984) [Crossref]
11. H. Hwang, R. Haddad, Adaptive median filters: new algorithms and results. IEEE Trans. Image Proc. **4**(4), 499–502 (1995) [ADS][Crossref]
12. L. Euler, *Introductio in Analysin Infinitorum (Tomus Primus)* (M.M. Bousquet, Lausanne, 1748)
13. A. Oppenheim, A. Willsky, S. Nawab, *Signals and Systems* (Pearson Education, 1997)
14. S. Chern, W. Chen, *Lectures on Differential Geometry (in Chinese)* (Peking University Press, 1983)
15. M. Spivak, *A Comprehensive Introduction to Differential Geometry* (Publish or Perish, 1999)

16. R. Gamkrelidze, D. Alekseevskij, V. Lychagin, A. Vinogradov (eds.) *Geometry I: Basic Ideas and Concepts of Differential Geometry* (Springer-Verlag, 1991)
17. J. Bonnans, J. Gilbert, C. Lemarechal, C. Sagastizabal, *Numerical Optimization: Theoretical and Practical Aspects* (Springer Science & Business Media, 2006)
18. A. Witkin. Scale-space filtering, in *International Joint Conference on Artificial Intelligence* (1983)
19. T. Lindeberg, Scale-space theory: a basic tool for analysing structures at different scales. J. Appl. Stat. **21**(2), 225–270 (1994) [Crossref]
20. J. Koenderink, The structure of images. Biol. Cybernet. **50**, 363–370 (1984) [MathSciNet][Crossref][zbMATH]
21. L. Evans, *Partial Differential Equations* (American Mathematical Society, 2010)
22. S. Lang. *Introduction to Linear Algebra* (Springer-Verlag, 1970)
23. D. Lay, *Linear Algebra and Its Applications* (Pearson Education, 2003)
24. S. Jain, A. Gunawardena, *Linear Algebra: An Interactive Approach* (Brooks/Cole, 2004)
25. A. Quarteroni, R. Sacco, F. Saleri, *Numerical Mathematics* (Springer Science & Business Media, 2000)
26. G. Stewart, *Matrix Algorithms: (Volume I) Basic Decompositions* (SIAM, 1998)
27. R.A. Horn, C.R. Johnson, *Matrix Analysis* (Cambridge University Press, 2012)
28. G. Stewart, *Matrix Algorithms: (Volume II) Eigensystems* (SIAM, 2001)
29. K. Levenberg, A method for the solution of certain non-linear problems in least squares. Quart. Appl. Math. **2**(2), 164–168 (1944) [MathSciNet][Crossref][zbMATH]
30. D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters. J. Soc. Ind. Appl. Math. **11**(2), 431–441 (1963) [MathSciNet][Crossref][zbMATH]

31. J. Moré, The levenberg-marquardt algorithm: implementation and theory, in *Numerical Analysis, Lecture Notes in Mathematics* ed. by G.A. Watson (1977) pp. 105–116
32. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev. **65**(6), 386–408 (1958) [Crossref][PubMed]
33. M. Minsky, S. Papert. *Perceptrons* (MIT Press, 1969)
34. D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors. Nature **323**, 533–536 (1986) [ADS][Crossref]
35. D. Rumelhart, G. Hinton, R. Williams, Learning internal representations by error propagation. Parallel Distributed Processing **1**, 318–362 (1986)
36. H. Li. On representation learning with feedback. ArXiv 2022
37. C. Chen, H. Li. Robust representation learning with feedback for single image deraining, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7742–7751 (2021)

Index

A
Abelian 270
Adaboost 118
  adaptive boosting 118
Adaptive thresholding 36–38, 296, 322, 370
ADAS 347
  advanced driver assistance system 347
Adjacency list 167
Adjacency matrix 167, 169, 315
Affine combination 176
Affine hull 177
Affine set 176, 177
All-element-one vector **1** 117, 170, 317, 321
Analog
  analog camera 2, 4
  analog electric circuit 4
  analog electric device 4
  analog image 4
  analog television 4
Angle of incidence 217
Angle of refraction 217, 218
ANN 461, 462, 466, 467, 471, 473
  artificial neural network 118–121, 461, 462, 464–468, 471–476
Anticommutative law 269
AprilTag 370, 375–379, 382, 383
äußere maß 161
Augmented matrix 450
Augmented reality 9, 12, 292, 351, 356, 393
Augmented vector 196, 197, 328, 332, 471
Auto-correlation 407–410
Autoencoder 123

B

Background 33
Back-propagation 122, 467, 468, 471–473, 475
BFS 147
  breadth-first search 147
Bijective 191, 192, 204, 207, 208, 210, 252, 263, 267–269, 288, 350, 404, 446
Binary categorization 33
Binocular triangulation 387, 388
Binormal vector 431
Bipartition 315, 316, 320, 322, 323, 325
Bird-eye view 211
Bit operations 109, 364
  eXclusive OR 109, 364, 462
  XOR 109, 364, 462
Bivariate function 17, 22, 49, 58, 69, 72, 145, 408, 430, 433
Black box 468
Boosting 118
Box-counting dimension 164
Box filter 101, 102, 398
Box-packing dimension 164
BRIEF 107
  binary robust independent elementary features 107
  pixel value comparison test 107
  sampling patterns 107
Brownian motion 410

C

Calculus of variations 412
Camera calibration 251
  camera intrinsic calibration 251, 270, 283
  camera-range sensor calibration 12, 292
  co-calibration 292
  image distortion calibration 276
  stereo camera calibration 12, 291, 292

Camera coordinates system 194, 195, 217, 218, 222, 230–232, 235, 246, 270, 292, 353, 361, 363, 379, 383, 388, 392
Camera model 2, 11, 187, 189, 193, 194, 214, 215, 245, 246, 280
  camera extrinsic parameter 196, 197, 208, 251, 271, 274, 348–350
  camera intrinsic matrix 196, 197, 206, 253, 255, 260, 271, 272, 290, 379, 383, 384
  camera intrinsic parameter 12, 196, 208, 247, 251, 253, 258, 260, 261, 271, 272, 274, 275, 280, 282, 283, 348, 350
  distortion-considered camera model 11, 215, 216, 221, 223, 224, 276
  distortion-free camera model 11, 214
  pinhole camera model 11, 12, 194, 214, 215, 221, 225, 227, 228, 247, 271, 283, 353
  principal axis 194
  principal point 194, 215, 221
Canny edge detection 55, 75, 295
Cantor set 158
Cardinality 163, 305
Cauchy criterion 155
Cauchy inequality 286
Central limit theorem 407
Ceva theorem 152
Chain rule 469
Characteristic polynomial 268, 424, 425
Chebyshev inequality 406
City-block distance 145–147, 150
Closest point problem 11, 181, 307
Clustering 7, 12, 295, 297, 299–301, 305–309, 311, 313–315, 322, 323, 325
  adaptive clustering 301
  agglomerative analysis 301
  clustering point 297, 298
  clusters 295–297, 299–301, 306, 325, 372

  direct connection 295–297
  divisive analysis 301
  hierarchical clustering 301
  inter-cluster variance 301
  intra-cluster variance 301
  K-clustering 297, 298
  K-means 297–301, 306, 307
  mean shift 301, 304–309, 311, 313, 314, 322, 325
  normalized cut 315, 316, 320, 322, 323, 325
  recursive clustering 322
  segmentation 311, 313, 314, 325, 326
CNN 118
  convolutional neural network 118–120
Color image 4, 34
  blue channel image 4
  blue channel value 34
  green channel image 4
  green channel value 34
  red channel image 4
  red channel value 34
Combinatorial optimization 317, 468
Commutator 270
  commutator subgroup 270
Complete graph 315
Computational geometry 11, 129, 173, 176
Concentration condition 302
Configuration space 336, 337
Congruent transformation 320
Conic curve 191, 192
Conic section 191
Constrained optimization problem 138, 320
Continued fractions 154
Control points 208, 251, 257, 271, 274, 277, 279, 280, 282–284, 287, 289, 350
Convex combination 176
Convex hull 176–179

Convex polygon 177, 178, 180, 181
Convex set 176, 177
Cooperative augmented reality 12, 356, 360
Cooperative perception 356
Corner 56, 58, 59, 104–106, 109, 110, 211, 212, 214, 251, 257, 271, 351, 376, 377
Countable 161, 162, 404
Covariance 114, 151, 327, 335, 406
Cramer rule 199
Cross product 236, 269
Cross-ratio 201, 202
Cross validation 282
Cumulative distribution function 405
Curse of dimensionality 337
Curvature 430–433, 435, 437–439, 441, 446
  curvature formula 431–436, 439
Curve 50, 156, 164, 295, 429, 430
  planar curve 431, 432, 435, 436
  quadratic curve 204
  three-dimensional curve 429, 430, 434, 436
  two-dimensional curve 429

D

Data association 370
Deep learning 112, 326
Deep neural network 118
Degenerate 67, 69, 70, 137, 139–141, 199, 203, 204, 229, 353, 354
Degradation problem 121
Delaunay triangulation 183
Desargues theorem 187, 189, 190, 193
DFS 54
  depth-first search 54, 295, 296
Diagonally dominant 170
Diagonal matrix of degrees 169
Diagonal matrix of weights 169, 317

Difference-of-Gaussian 76, 77
Differential geometry 429
  function curve 433
  function representation 429, 431
  implicit curve 436
  implicit representation 429, 430, 432, 435
  implicit surface 439
  parametric curve 433
  parametric representation 429, 430, 432, 434
  parametric surface 436
Diffusion equation 446
Digital 1
  digital camera 1–4
  digital device 2, 4
  digital image 2–4
  digital processing 417
  digital processor 1
Digraph 168
  directed graph 168
Dimensionality reduction 114
Dirac function 410, 447
Discrete optimization 320
Disparity 248
  disparity image 248
  disparity map 248
Distance 90, 91, 111, 129, 131–138, 143, 152–154, 157, 167, 181, 182, 218
Distortion 215
  distortion-considered camera model 11, 215, 216, 221, 223, 224, 276
  distortion-free camera model 11, 214
  distortion-free image coordinates 219, 225
  distortion parameters 220, 223, 276, 280, 283
  distortion rectification 8, 225, 227, 283
  image distortion model 220, 221, 223
  image distortion scaling factor 220, 222

- inverse image distortion model 226

Divide-and-conquer 14, 178, 182, 420, 421

Double thresholding 53–56, 295

E

Edge 47, 50–56, 59, 75, 82, 123
- strong edge 54, 57, 295
- weak edge 54, 57, 295

Element-wise 13
- element-wise division 470
- element-wise exponential function 122, 471
- element-wise operation 471
- element-wise product 57, 470

EM 337
- E-step 339, 340
- expectation maximization 337, 339, 341
- M-step 339, 340

Epipolar constraint 12, 230–233, 235, 238, 240, 241, 284, 289, 383, 392
- duality of epipolar lines 240
- epipolar constraint rectification 244
- epipolar lines 239
- epipolar points 238
- epipoles 238
- essential matrix 232, 235, 236, 288–291, 384, 391, 392
- fundamental matrix 231, 233–235, 283–285, 288, 392

Essential matrix 232, 236, 288, 384, 391

Euclidean distance 90, 130, 131, 134, 143–145, 147, 151, 302

Euclidean space 132, 429, 430

Euler 166, 425
- partial fraction decomposition 425

Euler formula 171–173

Event-triggered image 4

Expectation 338, 339, 343, 405–407, 409, 445

Expected value 405

Exponential decay factor 472
Extreme point 177, 178
Extremum
  extremum condition 40, 41, 217
  extremum point 40, 41, 444–446

F

FAST 104
  16-position full checking 105
  4-position partial checking 105
  features from accelerated segment test 104
Faster multiplication 14
Feature 7, 11, 12, 17, 32, 33, 40, 60, 76, 79, 98, 100, 101, 104, 107, 108, 113–116, 119, 123, 295, 297–301, 305–309, 311, 313, 315, 322, 323, 325, 326, 336–340, 342, 343, 365, 366, 387–389, 444, 462–464, 475, 476
  abstract feature points 11, 59
  feature descriptor 107, 111, 364, 365
  feature extraction 11, 17, 33, 59, 87, 92, 100, 101, 118, 123, 124
  feature map 119, 120
  feature-model relationship 12, 326, 335, 342
  feature organization 295
  feature set 32, 33, 43
  feature space 7, 90, 113, 301–304, 306, 307, 309, 311, 463, 464, 475, 476
  intuitive feature points 11, 59
  salient feature 61, 444
Fermat principle of least time 216
Fermat principle of refraction 216
Filter 59, 403
  box filter 101, 102, 398
  filtered image function 60, 71, 73, 76, 77, 444, 446, 448
  filtering kernel 59, 83, 87
  filtering scale 443, 444, 446
  filtering-scale parameter 60, 61, 97, 443, 444

  - Gaussian filter 101, 296, 416, 417, 419, 443
  - median filter 420, 423
  - optimal filter 59, 412, 443
  - parametric filter 59, 60, 443–445, 448
  - Wiener filter 59, 410, 412, 415–417, 424, 443
- First fundamental form 437
- Fish-eye camera 215
- Floor function 19
- Foreground 33
- Fourier transform 65, 415, 447
- Fractal 154, 156, 160, 374
  - box-counting dimension 164
  - box-packing dimension 164
  - Cantor fractal 158–160, 164, 165
  - fractal dimension 157, 160, 164, 165
  - fractal measure 152, 164, 165
  - fractal transform 156, 160
  - Γ shape fractal 165
  - Koch curve 156
  - non-scaling fractal 156
  - scaling fractal 156
  - self-affine transform 156
  - self-similar fractal 160
  - self-similar transform 154–156, 158, 159
  - snow fractal 156–158, 160, 164, 165
  - stochastic fractal transform 162
  - stochastic Koch curve 160
  - stochastic self-affine transform 156
  - stochastic self-similar transform 156
  - stochastic snow fractal 160–162, 164, 165
- Fractional polynomial 424
- Frenet formula 430, 434, 435, 437, 439, 445
- Frequency-domain analysis 424
  - magnitude 427
  - phase (phase angle) 427
- Frequency transfer function 427

Frobenius norm 284
Functional optimization 412, 413
Fundamental matrix 231, 283–285
Fundamental solution 447

G

GAN 123
  discriminator networks 123
  generative adversarial network 123
Gaussian elimination method 449, 450
Gaussian filter 296, 416, 417, 443
  Gaussian filtered images 61–64, 77, 79, 80, 89
  Gaussian filtering 50, 55, 57, 60, 63, 83, 87, 100, 101, 104, 105, 418–420
  Gaussian filtering kernel 83, 87
  one-dimensional Gaussian filter 417, 419
  parametric Gaussian filter 60, 61, 64, 65, 76, 101, 448
  special two-dimensional Gaussian filter 419, 420
  two-dimensional Gaussian filter 418–420
Gaussian pyramid 11, 61, 64, 80, 101
Generative model 337
Geodesic 132
Geometric series 61–64, 77, 79, 80
Gersgorin discs 170
Gersgorin theorem 170
GPU 120
  graphic processing units 120
Gradient image 47–56
  u-gradient image 47, 56, 57
  v-gradient image 47, 56, 57
Graph 166, 315
  dense 167
  edge 166, 172
  face 172
  induced sub-graph 315
  planar graph 171–173, 175, 184

  PSLG 173, 175, 177, 179
  simple graph 171
  sparse 167, 173, 175, 184
  vertex 166, 172
  weighted graph 315
Gray-level image 5, 32, 34, 401

H

Haar-wavelet 102, 123
  Haar basis 102
  Haar kernels 103
Hamadard division ⊘ 122, 470
Hamadard product ⊙ 57, 122, 470
Hamming distance 109, 365, 370
Harris corner detection 56, 58, 59, 104, 212
Harris image 58
Hausdorff dimension 162, 164
Hausdorff measure 160–162, 164
Heat equation 76, 446–448
Hessian matrix 64, 67, 69–73, 81, 101, 102, 439, 457, 458
  approximated Hessian matrix 458
  quasi-determinant 102
Heterogeneous 292
Hidden unit 118, 462–464, 466–468, 470, 473, 475, 476
Hidden variable 338
Histogram 5, 83–85, 87–89, 97, 423
  histogram matching 5
  histogram specification 5
Ho($\ddot{o}$)lder inequality 144, 145
Homeomorphic 171, 173
Homogeneous 222, 230
  homogeneous linear equation 209, 210, 230, 254, 255, 257, 258, 265, 266, 284, 289, 329, 376, 379, 382, 453, 455
Homography 2, 12, 124, 204, 207, 210, 212, 233, 241, 244, 252, 254, 255, 257, 258, 270, 271, 349, 364, 367, 368, 376,

377
  homography calibration 208, 210
  homography matrix 207–210, 212, 233, 243, 252, 365, 366, 376, 377, 379, 382, 453
Hough transform 335–337
  circle Hough transform 337
  line Hough transform 337
Hyperplane 115, 137

I

Ideal image coordinates 219, 225
Ill-conditioned 329, 382
Image compression 5, 165
Image coordinates system 11, 17, 18, 80–82, 88, 97, 194, 195, 197, 205, 217, 218, 238–240, 283, 289, 292, 347, 357, 358, 360, 361, 364, 376
  physical image coordinates system 194
Image encoding-decoding 5
Image enhancement 5
Image filtering 5, 50, 59, 403, 411, 417, 443
Image pixel value function 11, 17–20, 22, 24, 47, 52, 56, 60, 64, 395, 444, 448
Image processing 1, 2, 4, 5, 17, 18, 32, 52, 59, 60, 64, 280, 347, 370, 423, 443
Image reconstruction 5
Image resolution enhancement 30
Image restoration 5
Image stitching 9, 12, 97, 112, 124, 361, 363, 367, 369
Independent component analysis 114
Index set 407
Index shift 407, 408
Indicator variable 320, 338, 339
Indicator vector 317, 322, 338
Induced sub-graph 315
Infra-red image 4
Injective 263

Inner product 52, 374
Integral image 37, 101, 102, 109, 378, 395, 397, 398
  integral image iterative computation 396, 398
Internal representation 462
Interpolation 11
  bilinear interpolation 20
  biquadratic interpolation 24, 31, 32
  image resolution enhancement 11, 30–32
  linear interpolation 18–20, 214, 307, 311
  quadratic interpolation 18, 22, 24, 30, 58, 80–82, 85
  trilinear interpolation 87, 89
Inverse Laplace transform 424
Inverse perspective mapping 210, 211, 213, 214, 227, 350
Inverse rotation function 263, 349, 382, 384
Isomorphic 268
Isomorphism 269

J

Jacobian matrix 456
Jacobi identity 269
Joint 113, 116, 313, 327
Jordan measure 156, 161

K

Kalman filter 335
Karhunen—Loève Transform 114
Kernel 101, 113, 114, 118–121, 302, 306, 313, 462
  concentration condition 302
  filtering kernel 59, 83, 87
  Haar kernel 103
  kernel bandwidth 306, 307, 322
  normalization condition 302
  normal kernel 305
  profile function 303, 304
  radial symmetric kernel 302
  triangular kernel 305, 322

- unbiasedness condition 302
- uniformity condition 302
- uniform kernel 305

Kernel density estimator 302
Kernel estimation 301
Kernel space 235
K-means 297–301, 306, 307
K-nearest neighbours 113
Koch curve 156
Kronecker product 25, 29, 254, 284

L

La courbe de von Koch 156
Lagrange multiplier 138, 438, 439
Laplace transform 65, 424
- Laplace transform pairs 424

Laplace variable 424
Laplacian matrix 169, 317
Laplacian operator 64, 65, 71, 73, 446
- scale-invariant Laplacian layer 77–80
- scale-invariant Laplacian operator 71, 73, 74, 76

Learning rate 472
Least squares 327, 328, 457
Lebesgue measure 156, 161
Levenberg-Marquardt 275, 458
Lie group 266, 267, 269, 270
- Lie algebra 269
- tangent space 268, 269

Linear equation 14, 265, 277, 278, 319, 320, 328, 329, 332, 333, 388, 392, 449, 450, 452, 456
Linear programming 117
Linear time-invariant 411, 423, 424
LMS 326, 327, 332
- least mean squares 326–328, 335, 456, 457, 469
- recursive linear LMS 332
- regularized LMS 335

L_1-norm distance 90, 145
L_2-norm distance 144, 145
L_p-norm distance 143–145
L_∞-norm distance 145
Logistic activation function 120, 461, 469
Logistic sigmoid function 461
Loss function 116, 335
Low-pass filter 427
LSTM 121
  long short-term memory 121, 122
  memory blocks 122

M
Machine learning 112
Magnetic image 4
Mahalanobis distance 151, 302
Markov inequality 406
Matching 7, 87, 91, 92, 98, 112, 241, 242, 248, 364, 365
Mathematical induction 172
Matrix equation 237, 243
Matrix exponential function 267
Matrix logarithm function 267
Matrix vectorization 25, 254
Maximum likelihood 326, 327, 338
Maze-like polygon 179
Mean shift 301, 304–309, 311, 313, 314, 322, 325
  grid interpolation 311
  grid sampling 307–309, 311, 314
  grid sampling-interpolation 307, 311, 313, 314
Measure 11, 113, 143, 152–154, 156, 157, 161, 163–165
  area 152, 153, 157
  fractal measure 152, 164, 165
  Hausdorff measure 160–162, 164
  inner measure 161
  Jordan measure 156, 161
  Lebesgue measure 156, 161

  outer measure 161
  volume 152, 153, 161
Measurement 450, 451
  measurement matrix 335
Median filter 420, 423
Menelaus theorem 152, 153, 187, 188, 190, 192, 193
Minimum cut problem 317
Minimum-distance-path perspective 131, 132, 134, 135, 137
Minkowski dimension 164
Minkowski inequality 144
Mixture model 337–339
Mixture weight 337
Mode 303, 306, 307, 311, 313
Model fitting 12, 326, 327, 335, 342, 459
Moment 109, 405
Multi-digit number multiplication 14, 165
Multiplication 13, 14, 165
Multi-sensor fusion 292
  camera-range sensor fusion 292

N

Natural number 404
Nested intervals 154
Newton method 457, 458
  quasi-Newton method 457, 458
Non-integer dimension 160
Nonlinear equation 272–274, 276, 277, 291, 303, 331, 349, 455, 456, 459, 460
Nonlinear filtering 420
Non-maximum suppression 40–43, 51, 52, 58, 71, 77–79, 102, 105
  directional non-maximum suppression 50, 53, 55
Non-minimum suppression 374
Non-negative condition 140, 141, 353, 354, 404
Non-negative matrix factorization 114

Normalization condition 140, 141, 266, 273, 302, 337, 354, 404
Normalized cut 315, 316, 320, 322, 323, 325
  normalized cut measure 316, 322, 323, 325
Normal vector 353, 431–433, 436, 437, 439, 444
Normal view 8, 211, 212, 214
Numbered musical notation 34
Numeric 288, 458, 467
  numeric approximation 48, 76, 81
  numeric computation 113, 220, 306, 335, 337
  numeric method 48
  numeric operation 14
  numeric optimization 274, 335, 467, 468
  numeric singularity 13
  numeric stability 291

O

Objective function 298, 321
Obstacle distance map 145–150
Octave 61–64, 77–82, 87, 89
Offset 80, 81, 83, 218, 405
Optimality condition 138, 413, 414, 438, 439
Optimization 258, 270, 280, 297, 320, 322, 327, 328, 412, 452, 453, 455, 456, 458
ORB 109–112, 124, 283, 287, 364, 366, 368, 369, 391
  oriented FAST and rotated BRIEF 109
Orientation histogram 83–85, 87–89, 97
Over-fitting 120, 282, 476

P

Panorama 9, 124, 361, 369
Pappus theorem 187–189, 193
Parallax 248
Parametric 432, 434, 436
  parametric curve 433
  parametric filter 59, 60, 443–445, 448

  parametric function 327, 468
  parametric model 12, 326
  parametric nonlinear function 275, 277, 280, 459, 467, 468
  parametric representation 429, 430, 432, 434
  parametric surface 436
  parametric target model 328, 330, 332
  parametric variable 429, 430
Parametrize 134, 135, 137, 198, 200, 203, 222, 337, 339, 429, 430
Partial differential equation 446
  fundamental solution 447
Partial fraction decomposition 425
Partial sorting 106
Pascal theorem 187, 190–193
Pattern 295
  pattern recognition 1, 2, 7, 12, 295, 370
PCA 114
  principal axis 114
  principal component analysis 114, 145
Perspective mapping 188, 189, 197, 198, 203, 204, 210
  perspective mapping matrix 197, 198, 204
Perspective transform 188
Photosensitive plane 2, 194, 195
Pinhole camera model 11, 12, 194, 214, 215, 221, 225, 227, 228, 247, 271, 283, 353
Pixel 3
  pixel value 3
  pixel-wise 33, 57
  space discretization 2, 3
  value discretization 2–4
Point-inclusion problem 179
Point-location problem 175
Poisson bracket 269, 270
Polygon-sector method 181
Polynomial multiplication 14

Power series 219, 343, 344, 426
Principal axis 114, 194
Principal curvature 438, 439
Principal direction 114
Principal point 194, 215, 221
Probability density function 301, 404, 405
Probability mass function 404, 405
Profile 302
Projective geometry 2, 11, 187–191, 193, 198, 201, 233
  conic curve 191, 192
  conic section 191
  invariance in projective geometry 188, 192, 224
PSLG 173–175, 177, 179
  doubly-connected edge list representation 174
  planar straight-line graph 11, 168, 173–175, 177, 183, 184
  winged-edge representation 174, 175

Q

Quadratic programming 117, 452
Quadrilateral 372, 374–376
Quantization 2
Quicksort 420
Quotient group 270

R

Random variable 403–408
  continuous random variable 404, 405
  discrete random variable 404
  independent random variable 407, 409
  offset random variable 405
RANSAC 287, 341, 342, 366, 389, 391
  RANdom SAmple Consensus 342, 366, 389
Rayleigh quotient 454
Rayleigh ratio 114, 454
Recursive estimation 335

Refraction 215, 216, 218, 221
Registration 87, 255, 257
Regularization 335
Relativity theory 132
ReLU 120, 462
  rectified linear unit 120, 462
RGB 4
  red-green-blue 4
Rhombus 162
Ridge regression 335
Riemannian space 132
Right-angle triangle 152
  Gou Gu theorem 152
  Pythagorean theorem 152
RNN 121
  recurrent neural network 121–123
  vanishing gradient problem 122
Rodrigues formula 263, 265, 266
Rotation function 263, 273, 349, 382, 386
Round function 22

S

Sample space 403
Sampling 2
Scale-space 60–62, 64, 71, 73, 76–80, 82, 88, 444
  discrete scale space 71
  filtered image function 60, 71, 73, 76, 77, 444, 446, 448
  filtering-scale parameter 60, 61, 97, 443, 444
  Gaussian filtered images 61–64, 77, 79, 80, 89
  octave 61–64, 77–82, 87, 89
  scale-space analysis 60, 100, 448
  scale-space discretization 76
  scale-space feature extraction 59, 80, 101
  scale-space features 79
Scale-space theory 11, 59, 443, 444
  causality 60, 444, 448

  - homogeneity 60, 445, 448
  - isotropy 60, 445, 448
- S-domain function 424
- Second fundamental form 438
- See-through 356, 360, 361
- Segmentation 311, 313, 314, 325, 326
  - instance segmentation 326
  - semantic segmentation 326
- Sequence learning 121
- Serret-Frenet formula 430
- Seven bridges of Konigsberg 166
- Shift-invariant 119, 408, 410
- SIFT 11, 79–92, 97–102, 104, 119, 124, 283, 287, 356, 364, 366, 368, 369, 391–393
  - feature point localization 79, 80
  - local image descriptor 79, 87–90, 92, 97, 98
  - local image orientation assignment 79, 82, 87
  - scale invariant feature transform 11, 79, 97, 98
- Signal filtering 403, 417
- Signal-noise model 408, 410
- Signal processing 408, 410, 411
- Simplex method 117
- Singular value decomposition 237, 242, 284, 290, 384, 392
- Skew-symmetric 267, 435
  - skew-symmetric matrix 236, 262, 267–269, 385, 435
- SLAM 379
  - simultaneous localization and mapping 8, 12, 379, 383
  - visual SLAM 8, 9, 12, 97, 383, 384
- Special orthogonal group 266, 268, 269
- Standard deviation 342, 344, 443
- State 335, 427, 463, 476
  - state covariance 335
  - state estimation 118
  - state vector 335
- Stationary point 444
- Steepest descent method 458

Stereo vision 12, 227–229
  binocular disparity 12, 245, 248
  binocular vision 227
  depth 248
  disparity 248
  ideal stereo vision 245
  multi-view stereo vision 292
  parallax 248
  stereo baseline 247
Stochastic process 403, 407–409
  shift-invariant stochastic process 408, 410
Strong law of large numbers 407–409
SURF 101
  speeded up robust features 101
Surface 22, 132, 352, 429, 430, 436–439, 444, 445
  Gaussian curvature 439, 441
  ground surface 248, 347–351
  maximum curvature 439, 440, 444, 445
  mean curvature 439, 441
  minimum curvature 439, 440, 444, 445
  principal curvature 438, 439
  sphere surface 441
  surface differential 437
  three-dimensional surface 430
Surjective 263
SVM 114
  hard margin 115
  soft error 116
  soft margin 116
  support vector machine 114, 115, 118, 462
Symbolic operation 253, 268

T

Tag 12, 370
  AprilTag 370, 375–379, 382, 383
  tag decoding 376, 377

  tag detection 378, 379, 382
  tag map 379
Tangent plane 436
Tangent space 268, 269
Tangent vector 431, 432, 434–437, 439
Tensor 42, 43, 47, 119, 120
Tetrahedron 9, 171, 183, 190, 351, 352, 355
Thresholding 33, 34, 36, 38, 53, 54, 56, 58, 322
  background 33, 34, 37, 38, 43, 44, 75
  double thresholding 53–56, 295
  foreground 33, 34, 36–38, 43–45, 295, 296, 370, 374
  strong thresholding 55
  weak thresholding 55
Time-domain function 424, 426
Time-invariant 251, 411, 423, 424
Torsion 431
Training randomness 475
Transfer function 424
Triangular inequality 131–133, 144, 151, 152
Trivariate function 42, 70, 72, 73, 444
Two-cluster decomposition 315

U
Unbiasedness condition 302
Uncountable 61, 336, 404
Undirected graph 168
Uniformity condition 302
Univariate function 19, 22, 69, 72, 145

V
Value assignment 148, 244, 322
Value space 403–406
Vanishing point 189, 200
Variance 114, 343, 405–407, 409
Vehicle-ground coordinates system 347, 348, 350, 351, 358, 359

Virtual file scanning 8, 211, 213
Virtual reality 9, 12, 351, 352, 355, 356
Virtual vertex at infinity 172, 173, 183
Vision discretization 2, 17
Vision systems 1
  biological vision systems 1, 38, 47, 60, 119, 165, 295, 401
  computer vision systems 1, 47, 165, 295
Visual localization 12, 370, 379, 383
Visual SLAM 8, 9, 12, 97, 383, 384
Voronoi diagram 182–184
  paralléloèdres 183
  sommet primitif 183
  sommet simple 183
  Voronoi edges 183
  Voronoi vertices 183

W

Watermark removal 34, 36
White box 468
White noise 409, 410, 414
  white noise condition 409
  white noise energy 416
Wide-angle camera 8, 12, 215
Wiener filter 412, 417
  frequency domain Wiener filter 415
World coordinates system 194, 195, 197, 201, 203, 205, 230–233, 246, 251, 255, 257, 270, 271, 347, 353, 361